HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

8-28-00

A

PATENT APPLICATION

ATTORNEY DOCKET NO. 10003284-1

08/23/00 U.S. PTO

jc896 U.S. PTO 09/645790 08/24/00

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): (X) Utility ( ) Design
(X) original patent application,
( ) continuation-in-part application

INVENTOR(S): **Zhen He et al**

TITLE: **Combined Dot Density And Dot Size Modulation**

Enclosed are:

(X) The Declaration and Power of Attorney. ( ) signed (X) unsigned or partially signed

(X) 10 sheets of drawings (one set) ( ) Associate Power of Attorney

( ) Form PTO-1449 ( ) Information Disclosure Statement and Form PTO-1449

( ) Priority document(s) (X) (Other) **Appendices A1, A2, B1, B2 & B3** (fee $_____)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	38 — 20	18	X $18	$ 324
INDEPENDENT CLAIMS	6 — 3	3	X $78	$ 234
ANY MULTIPLE DEPENDENT CLAIMS	0		$260	$ 0
		BASIC FEE: Design ($310.00); Utility ($690.00)		$ 690
			TOTAL FILING FEE	$ 1,248
			OTHER FEES	$
			TOTAL CHARGES TO DEPOSIT ACCOUNT	$ 1,248

Charge $ 1,248 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16, 1.17,1.19, 1.20 and 1.21. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EH862490456US

Date of Deposit 8/23/00

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

By Betty Hinkle

Typed Name: Betty Hinkle

Respectfully submitted,

Zhen He et al

By _____

Steven R. Ormiston

Attorney/Agent for Applicant(s)

Reg. No. 35,974

Date: 8/23/00

Telephone No.: (208) 396-2544

Rev 05/00 (TransNew)

- Attach as First Page to Transmitted Papers -

"Express Mail" mailing label number: EH862490456US
Date of Deposit: 08/23/00
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" services under 37 C.F.R. 1.10 on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.
Typed Name of Person Mailing Paper or Fee: Betty Hinkle
Signature: Betty Hinkle

PATENT APPLICATION
DOCKET NO. 10003284-1

COMBINED DOT DENSITY AND DOT SIZE MODULATION

INVENTORS:

Zhen He
Charles A. Bouman
Qian Lin

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A. Bouman
Qian Lin

CROSS REFERENCE TO APPENDICES

This application is related to Application Attorney Docket No. 10001606-1 which is incorporated herein by reference in its entirety.

CROSS REFERENCE TO APPENDICES

Appendix A1, which is part of the present disclosure, is an appendix consisting of 8 pages. Appendix A1 lists source code written in C programming language of an illustrative embodiment of the present invention using an eight-bit code to control a printer system having a pulse width modulated laser capability for each printed pixel. Appendix A2, which is part of the present disclosure, is an appendix consisting of 13 pages. Appendix A2 lists the particular coefficients used in the Tone-Dependent Error Diffusion process for the eight-bit code used in accordance with an embodiment of the present invention. Appendix B1, which is part of the present disclosure, is an appendix consisting of 14 pages. Appendix B1 lists source code written in C programming language of an illustrative embodiment of the present invention using a two-bit code to control a printer system having a pulse width modulated laser capability for each printed pixel. Appendix B2, which is part of the present disclosure, is an appendix consisting of 2 pages. Appendix B2 lists source code written in C programming language that is used in conjunction with Appendix B1 for a two-bit code used to control a printer system having a pulse width modulated laser capability for each printed pixel. Appendix B3, which is part of the present disclosure, is an appendix consisting of 22 pages. Appendix B3 lists the particular coefficients used in the Tone-Dependent Error Diffusion process for the two-bit code used in accordance with an embodiment of the present invention.

COPYRIGHT NOTICE

reproduction by anyone of the patent document or the patent disclosure, as it appears in the patent and trademark office patent file or records, but otherwise reserves all copyright rights whatsoever.

FIELD OF THE INVENTION

The present invention relates to halftoning, and in particular to a halftoning method that combines dot size modulation and dot spacing modulation to control the overall gray level in an image forming or printing device.

BACKGROUND

Continuous-tone images, such as charts, drawings, and pictures, may be represented as a two-dimensional matrix of picture elements (pixels). The spatial resolution and intensity level for each pixel are chosen to correspond to the particular output device used. Typically, digital halftoning is used to transform a continuous-tone image into the desired matrix of pixels.

Conventional methods for digital halftoning generally fall into two categories: clustered dot and dispersed dot. As is well known in the art, in clustered dot techniques, the size of the printed dot is varied to control the perceived gray level or, equivalently, density of the printed tone. These methods may be thought of as amplitude modulation (AM) halftoning techniques since the amplitude or size of the dots controls the printed gray level.

Fig. 1 is a block diagram illustrating a conventional clustered dot halftoning process. As shown in Fig. 1, typically, an image x(m, n) is transformed, or tone compensated, at block 2, so that the printed gray level for each input value is correct. Tone compensation (block 2) is used to correct for attributes of the halftoning algorithm, image calibration, and printer behavior. In particular, most printers are not linear due to the inevitable effects of dot overlap. The dot size is then modulated at block 4 and the resulting signal is sent to the printing system. Typically, dot size modulation is performed using screening, which uses a threshold array that is compared to the pixel value. AM halftoning methods have certain advantages such as increased robustness to printing artifacts. However, a limitation of AM halftoning methods is that the choice of dot size presents a fundamental tradeoff between spatial resolution and number of gray levels.

Dispersed dot halftoning methods control the printed gray level through the spacing or, equivalently, the frequency of dot placement. Dispersed dot halftoning may be thought of as a frequency modulation (FM) halftoning technique since the frequency or spacing of the dots controls the printed gray level. Dispersed dot halftoning methods are well known, and include, for example, error diffusion halftoning, screening and, most recently, iterative search based halftoning. Fig. 2 is a block diagram illustrating a conventional dispersed dot halftoning process. As with the clustered dot halftoning process, the image x(m, n) is transformed, or tone compensated at block 6, so that the printed gray level for each input value is correct. The spacing of the dots is then appropriately dispersed at block 8 and the resulting signal is sent to the printing system.

Typically, dispersed dot halftoning uses the smallest dots possible to print as it is undesirable to actually notice the dots in the printed image. However, a drawback of dispersed dot halftoning is that the printer must be capable of printing well formed isolated dots. For example, error diffusion is typically used only in ink jet type printers because they generally can print stable isolated dots. However, error diffusion is typically not used in commercial electrophotographic printers and copiers, such as laser printers, due to their instability in producing binary or multilevel halftones.

SUMMARY

A halftoning system, in accordance with an embodiment of the present invention, uses dispersed dot halftoning in conjunction with dot size modulation to produce an image in which the density and size of the dots are modulated in conjunction to control the overall gray level in an image forming system. The dot density is modulated to control the spacing or frequency of the dot to be printed for a particular pixel location with respect to dots to be printed at preceding and subsequent pixel locations, while the dot size is modulated to control the size of the dot to be printed at the particular pixel location. The dot density and size modulation system may be used with an image forming or printing system, e.g., a computer and a printer. It should be understood that the image forming or printing system of the present invention is intended to include, but is not limited to, an electrophotographic printer

or copier, an ink jet printer, a facsimile machine or any other device that may be used to print an image. The dot density and size modulation system is particularly useful in modern electrophotographic printing systems that allow the printed dot size to be almost continuously varied through the specification of a pulse width modulation (PWM) code.

In accordance with an embodiment of the present invention, the dot density is modulated by controlling the dot density for a pixel location using an input pixel value for that pixel location and by performing dispersed dot halftoning to produce a dot position based on the dot density. For example, a look up table may generate a dot density value based on the input pixel value, which is then used in the dispersed dot halftoning process. The dispersed dot halftoning process may be error diffusion, e.g., tone dependent error diffusion, dispersed dot screening, or iterative search based halftoning. Tone dependent error diffusion is particularly advantageous because it generates high quality dispersed dot patterns and is computationally efficient.

The dot size is modulated, in accordance with an embodiment of the present invention, by independently controlling the dot size for the pixel location using the input pixel value and performing dot size modulation based on both the dot size and the dot position. Thus, for example, a separate look up table may be used to generate the dot size value based on the input pixel value. Advantageously, the dot size control and dot density control may be precomputed, and in one embodiment are designed with respect to each other and the characteristics of the printing system to optimize the image while matching the printing device characteristics. The resulting dot size value, as well as the dot position from the dispersed dot halftoning process, are then used to generate a signal, e.g., a pulse width modulated code, that controls the printer system. The signal may be produced by another look up table and may include information regarding both the pulse width of the desired dot for the pixel location as well as the justification, e.g. left, center, or right, for the pixel location.

In accordance with an embodiment of the present invention, the dot size look-up table and dot density look-up table are designed with respect to each other and the characteristics of the printing system in a closed loop measurement process to optimize the quality of the image while matching the printing device characteristics.

The system, in accordance with the present invention, uses a combination of the dot size and dot density to produce the desired gray level, which offers advantages over pure dot density modulation systems or pure dot size modulation systems. The combination of dot density and dot size modulation allows an extra degree of flexibility, which can be used to increase the visual quality of the halftoned pattern and/or increase the robustness of the halftoning to printer artifacts and variations.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram illustrating a conventional clustered dot halftoning process.

Fig. 2 is a block diagram illustrating a conventional dispersed dot halftoning process.

Fig. 3 is a block diagram illustrating a dot density and dot size modulation (referred to herein as AM/FM halftoning) system, in accordance with an embodiment of the present invention.

Fig. 4 is a schematic diagram of a printing system, including a printer and a computer that may be used with the present invention.

Fig. 5 shows a block diagram of an embodiment of the AM/FM halftoning system in which the dot density control and dot size control are look-up tables and the dispersed dot halftoning is tone dependent error diffusion.

Fig. 6 shows a block diagram of a tone dependent error diffusion, which may be used in the AM/FM halftoning system of Fig. 5.

Figs. 7A and 7B show an array of pixels and a pixel pair, respectively, illustrating the error diffusion used in accordance with an embodiment of the present invention.

Figs. 8A, 8B, and 8C show an idealized laser modulation output for a PWM system in which the width of the pulse may be less than the full width of the pixel and may be left, center, or right justified, respectively.

Fig. 9 illustrates the method used to form a circular dot with an area $\theta = 2$ and that is centered at pixel (m,n).

Fig. 10 is a flow chart showing a method of optimizing the dot size and dot density look-up tables using a closed loop measurement process in accordance with an embodiment of the present invention.

Fig. 11 is a graph showing the measured tone curve, $g_\theta(n)$, i.e., the output absorptance, versus input dot density for a fixed value of dot size $\theta = 1$, where a square dot shape is used.

Fig. 12 is a graph showing the print distortion, $D_\theta(n)$ versus input dot density for a fixed value of dot size $\theta = 1$, where a square dot shape is used.

Fig. 13 is a graph showing a set of inverse tone curves for dot sizes ranging from 0.3 to 1.4.

Fig. 14 is a graph showing a set of plots of the print distortion versus the desired output absorptance for dot sizes ranging from 0.3 to 1.4, where each curve corresponds to $D_{\theta,g}$ as a function of g for fixed θ.

Fig. 15 is a graph showing a dot size curve for different σ values, where the optimized dot size look-up table, θ_i used is $\sigma = 0.05$.

Fig. 16 is a graph showing a dot density curve for an optimized dot density look-up table that corresponds to the curve $\sigma = 0.05$ shown in graph in Fig. 15 and is a function of the desired absorptance.

Fig. 17 is a graph showing another dot size curve for an optimized dot size look-up table as a function of desired absorptance.

Fig. 18 is a graph showing a dot density curve for an optimized dot density look-up table that corresponds to the curve shown in graph in Fig. 17 and is a function of the desired absorptance.

Fig. 19 shows another embodiment of the AM/FM halftoning system, in accordance with the present invention.

Fig. 20 shows a block diagram illustrating another embodiment of the AM/FM halftoning system.

DETAILED DESCRIPTION

Fig. 3 is a block diagram illustrating a dot size and density modulation system (referred to herein as an AM/FM halftoning system) 100, in accordance with an embodiment of the present invention. As shown in Fig. 3, each pixel x(m,n) of an image to be printed is processed to determine the dot density, at dot density control 102, and the dot size, at dot size control 104. The AM/FM halftoning system 100 includes a subsystem 106 with a FM modulation portion, , e.g., dispersed dot

halftoning unit 108, and an AM modulation portion, e.g., dot size modulation unit 110.

The dot density control 102 provides a dot density value that is received by the dispersed dot halftoning unit 108 to determine the positions of dots on the discrete printing grid. The dot size control 104 provides a dot size value that is received by the dot size modulation unit 110 along with the positions of the dots from the dispersed dot halftoning unit 108. The dot size modulation unit 110 generates an output signal indicating the modulated size and modulated density of the dot to be printed for the input pixel x(m,n) by the printing system.

Fig. 4 is a schematic diagram of a printing system 112 that includes a host computer 113, a monitor 114 (e.g., a CRT), and a printer 116, which may be used with the present invention. Printer 116 could be any type of printer that prints black and/or color dots, including an electrophotographic printer, e.g., a laser printer, or copier, or an inkjet printer. Printer 116 typically includes a toner cartridge 117, which delivers particles of dry toner via electrostatic attraction to a rotating drum or sheet of paper. The paper is then heated to melt the toner so that the toner permanently adheres to the paper. For inkjet printers, a print cartridge is typically used that scans across the paper and prints droplets of ink in a well known fashion. Additional toner cartridges 118 may also be included in printer 116, for example, to provide different color inks. Printer 116 often includes a printer controller 119 for controlling the printing of dots, and in fact, may include the host computer 113 within printer 116, in which case monitor 114 is not needed. There are many well-known types of these devices, and details of their operation need not be presented here for a clear understanding of the operation of the present invention.

Fig. 5 shows a block diagram of one embodiment of an AM/FM halftoning system 120 in accordance with the present invention, in which the dot density control is a first 256x7 bit look-up table ("LUT") 122 and dot size control is a second 256x7 bit LUT 124. In this embodiment, the dot density control LUT 122 and dot size control LUT 124 map each input pixel x(m, n) to a specific desired value. In one embodiment, LUT 122 produces a dot density value ranging from 0 to 128 and LUT 124 produces a dot size value ranging from 0 to 63, but these values may vary if desired. For example, in some printing systems the dot density value may vary

between 0 and 255. Moreover, the dot density control and the dot size control, in other embodiments, may be more general and may depend on the local neighborhood of the input pixel x(m, n), which might be useful, e.g., for purposes of enhancing the quality of text or image edges, or more accurately rendering textures or smooth gradients.

Advantageously, both the dot density and dot size are varied independently based on attributes of the input image x(m, n). Nevertheless, neither the dot density control nor the dot size control independently control the printed tone. Instead, the printed tone is controlled through a combination of the two. The dot density and size are designed with respect to each other and the printing system to optimize the printed image while matching the device characteristics of the printing system. Consequently, tone correction may be incorporated directly into the combined density/size control.

As shown in Fig. 5, the FM modulation the dispersed dot halftoning unit is a tone dependent error diffusion (TDED) system 128. The TDED system 128 may be, for example, similar to that described in U.S. Patent Serial No.09/307,003, entitled "Tone Dependent Error Diffusion" to Pingshan Li and Jan P. Allebach, filed May 7, 1999, which is incorporated herein by reference. Tone dependent error diffusion is advantageous because it generates high quality dispersed dot patterns and is computationally efficient. The dispersed dot halftoning unit, however, may depend on the particular application, i.e., printer system, and may be any one of a wide variety of known halftoning methods including error diffusion, dispersed dot screening, or iterative search based halftoning such as direct binary search (DBS), which is described, for example, in M. Analoui and J. Allebach, "Model-Based Halftoning Using Direct Binary Search,", Proc. SPIE/IS&T's symp., electr, imag. sci. and tech., vol. 1666, pp. 96-108, San Jose, CA, Feb., 1992, and which is incorporated herein by reference.

Typically, the quantity varied in dispersed dot halftoning is the dot spacing, which is specified using the number of dots per unit area or dot density. It should be understood, however, that for some dispersed dot halftoning methods the output of the dot density control may be only approximately equal to the true dot density. For more information regarding dispersed dot halftoning in general, see for example, J.

Mulligan and A. Ahumada, Jr., "Principled Halftoning Based on Models of Human Vision," Proc. SPIE/IS&T's symp., electr, imag. sci. and tech., vol. 1666, pp. 109-121, San Jose, CA, Feb., 1992; and T. Pappas, and D. Neuhoff, "Least-Squares Model-Based Halftoning," Proc. SPIE/IS&T's symp., electr, imag. sci. and tech., vol. 1666, pp. 165-176, San Jose, CA, Feb., 1992, which are incorporated herein by reference.

Fig. 6 shows a block diagram of a TDED system 140, which may be used as the TDED system 128 of Fig. 5 to produce a TDED output in the form of a halftone value, in accordance with an embodiment of the present invention. The input pixel value, which has continuous-tone value, e.g., from 0 to 128, is represented as X(m,n), where m and n are the pixel locations. The TDED system 140 uses an error diffusion filter 142, which acts as a weighting matrix, and a threshold matrix 144.

The input pixel value is placed in a weight LUT 146, which may be, e.g., a 3x129x8 bit LUT, and which produces three different weights for the error diffusion filter 142. The input pixel value is also placed in a threshold weight LUT 148, which may be, e.g., a 2x129x8 bit LUT, and which produces two different thresholds for the threshold matrix 144. In addition, the pixel location (m,n) is received by a direct binary search screen (DBS) 150, the output of which is multiplied by the resulting output of threshold weight LUT 148, which is then summed with the other resulting output of threshold weight LUT 148. The summed result is then provided to the threshold matrix 144.

The error diffusion filter 142 provides an error value from previously processed pixels that is summed with the input pixel value X(m,n), to produce a modified pixel value u(m,n). The modified pixel value u(m,n) is received by the threshold matrix 144 and compared to at least one threshold level to produce the output halftone value g(m,n). A quantizer error value d(m,n) is produced as the difference between the output halftone value g(m,n) and the modified pixel value u(m,n). The quantizer error d(m,n) is received by the error diffusion filter 142, and is diffused to neighboring, subsequently processed pixel locations. See U.S. Patent Serial No. 09/307,003, entitled "Tone Dependent Error Diffusion" to Pingshan Li and Jan P. Allebach, filed May 7, 1999, which is incorporated herein by reference for a discussion of the implementation of the TDED system 140. As one of ordinary skill

in the art understands, a TDED system may be optimized for particular printing systems and printing characteristics by altering the particular process coefficient. For the particular coefficients used in accordance with an embodiment of the present invention, see Appendix A2.

Figs. 7A and 7B show an array of pixels 160 and a pixel pair 164, respectively, illustrating the error diffusion of the TDED system 128 in accordance with one embodiment of the present invention. As illustrated in Fig. 7A, the dots are printed using a serpentine raster scan as indicated by arrows 161. The pixels are organized in pixel pairs, in which a modulated dot will be fired in only one pixel illustrated with an "X". The pixels which are illustrated with an "O" will not have a dot fired. The dots are constrained to fire at positions on a diagonal grid pattern. The tone dependent error diffusion uses four weights that are diffused to four adjacent pixels, as indicated by arrows 162 and 163. As shown in Fig. 7A, the error is diffused to the next pixel to be processed, regardless of whether a dot will be fired in that pixel and to the three adjacent pixels in the next row, again regardless of whether a dot will be fired in that pixel. As shown in Fig. 7B, which shows a pixel pair 164, in one embodiment of the present invention, each pixel in a pixel pair 164 is justified towards the other pixel. Thus, the left hand pixel "X" is right justified and the right hand pixel "O" is left hand justified.

As shown in Fig. 5, the dot size modulation is a pulse width modulation (PWM) system 130 that directly modulates the size of the physical dot, as is well known in the art. The dot size modulation step requires very little computation since the appropriate PWM codes for each dot size may be precomputed and stored in a LUT and is accessed using the halftone value from the TDED system 128 and the dot size value from the dot size LUT 124. The dot size modulation, however, may also depend on the particular application, i.e., printing system, and may be any one of a number of known methods which vary the size of a printed dot either by grouping clusters of dots together, including clustered dot digital halftoning.

The PWM system 130 produces a signal, e.g., an eight-bit PWM code, that controls the size of the printed dot, for example, produced by a laser in an electrophotographic printer. As is well understood in the art, in an electrophotographic printer, the toner is only deposited when the laser is on. Thus,

each six-bit PWM code controls the width of a single pixel. The output signal from the PWM system 130 may also include two bits of information regarding the justification of the dot, i.e., whether the dot is left, center or right justified within the pixel.

The source code in Appendix A1 written in C programming language is one example of an implementation of the present invention using an eight-bit PWM code to control a printer system having a pulse width modulated laser capability for each printed pixel.

In Fig. 7A each grouping of two adjacent pixels is referred to as a pixel pair and is shown with an X on the left pixel and a 0 on the right pixel. A single example pixel pair is indicated at the bottom of Fig. 7. Each pixel pair is either enabled or disable depending on the TDED output value at the X position corresponding to the left hand pixel in the pixel pair. If TDED output value indicates that a dot should fire at the X position, then the pair is enabled. If the TDED output value does not indicate that a dot should fire at the X position, then the pair is not enabled. If the pair is not enabled, then the PWM codes for both the X position and the 0 position of the pixel pair are set to 0 and nothing is printed at either of the two pixels. If the pixel pair is enabled, then each of the two pixels are AM modulated separately using dot size value from the dot size LUT 124. More specifically, the PWM code for the left pixel, denoted by XPWM, and the PWM code for the right pixel, denoted by OPWM, are computed as follows:

XPWM =DotSizeLUT(left pixel)>>1+RightHandJustification; and equ. 1

OPWM =(DotSizeLUT(right pixel)+1)>>1+LeftHandJustification. equ. 2

If a dot is not fired, then XPWM=OPWM=off. Here the values "left pixel" and "right pixel" denote the values of the continuous tone image to be printed at the corresponding left and right locations of the pixel pair.

Figs. 8A, 8B, and 8C shows an idealized laser modulation output for a PWM system in which the width of the pulse may be less than the full width of the pixel and may be left, center, or right justified, respectively. Thus, for example, the lower six bits of a eight-bit PWM code may specify the width of the laser pulse, and the highest two bits specify the justification (left, center, or right). The pulse width value ranges

from 0 to 63, where 0 specifies a width 0 pulse, and 63 specifies a pulse that is the full width of the pixel. The PWM codes that may be used to specify the justification for each modulated pixel are given in Table 1.

1st Bit	2nd Bit	Value	Justification
1	1	0xc0	Right
0	1	0x40	Left
0	0	0x00	Center
1	0	0x80	Undefined

Table 1

Fig. 9 illustrates another method to form a dot, in accordance with an embodiment of the present invention. Fig. 9 shows a method used to form a circular dot 170 with an area $\theta = 2$ and that is centered at pixel (m,n). First, the radius of the desired ideal dot is determined as $r = \sqrt{\theta/\pi}$. Then, for each row of pixels, the area of intersection between the row and the ideal dot is calculated. To do this, let k index the rows of the image, and let $k = 0$ for the row containing the pixel (m, n). Then the region of intersection is defined by

$$S_k = \{(x,y): x^2 + y^2 < r \text{ and } | x - k | < 1/2\} \qquad \text{equ. 3}$$

where x and y are the distance from the center of the pixel (m, n). The area of intersection is then $\beta_k = \text{Area}\{S_k\}$. If $\beta_k > 1$, then $n_k = 2\lfloor(\beta_k - 1)/2\rfloor + 1$ pixels in the k^{th} row are fully turned on. For the remaining fractional area $\gamma_k = \beta_k - n_k$, two pixels on either side of the fully turned on pixels are turned on, each with area $\gamma_k/2$. The group of pixels are justified so that the dot is more compact, i.e., the left side pixel is right justified and the right side pixel is left justified so that the dot is formed by one pulse. If $\beta_k < 1$, we just turn on one pixel of area β_k with center justification. This is illustrated in Fig. 9 as array 172. The PWM codes, with values ranging from 0 to 63 and an indication of left (L), right (R), and center (C) justification, for each pixel is shown in array 174 in Fig. 9. Importantly, the mapping of the dot size θ to the PWM codes can be precomputed and implemented using a LUT in PWM 130 (shown in Fig.

5). Precomputing the PWM codes and using a LUT increases the speed of operation. In some embodiments, a rectangular dot rather than a circular dot may be used. The use of a rectangular dot has the advantage that when the dot area θ is less than 1 only a single pixel is modulated. The use of a rectangular dot makes the dot printing more stable for dot size $\theta > 1$.

When the dot area is greater than 1, i.e. θ is greater than 1, dot overlap may occur. In this case, the pulse widths are added, and only the first pulse justification is retained. More specifically, suppose there are N dots indexed by $i = 1,\cdots, N$ that correspond to the same pixel location. Let b_i be the justification part (the two most significant bits) of the PWM code, and let a_i be the pulse width part (the remaining six bits) of an eight-bit PWM code. Then the PWM code for a pixel is determined by

$$C_{m,n} = b_1 + \min(63, \sum_{i=1}^{N} a_i) \qquad \text{equ. 4}$$

The value of the pulse width is clipped if it exceeds the maximum value of 63. The effects of this clipping on the tone curve are removed when the dot density control LUT 122 (Fig. 5) and dot size control LUT (124) are designed to prevent distortion of the tone curve.

In some cases, 63 pulse width levels of PWM may be available or less than 63 levels of pulse width modulation may be available, which may cause contouring artifacts in the printed images. In either case, contouring happens because a change of one level in the PWM code is substantial enough to cause a visible difference in gray level. To eliminate this problem, the pulse width codes may be dithered. The dithering may be done using random numbers, a white noise threshold mask, a dispersed dot threshold mask (such as the DBS screen 150 shown in Fig. 6), or an error diffusion process. The dithering functions similar to multilevel halftoning algorithm and eliminates visual contours.

The dot density and dot size controls are selected to obtain the desired gray level, e.g. tone curve, for each input value x(m, n). Advantageously, the dot size control and dot density control may be precomputed, and in one embodiment are designed with respect to each other and the characteristics of the printing system in a closed loop measurement process to optimize the quality of the image while matching

the printing device characteristics. Because the tone curve is controlled through the combination of the dot density control and dot size control there is an additional degree of freedom that may be used to optimize a variety of printing attributes including print quality and/or print stability. In other words, it is possible to achieve most desired tone curves by either varying the size of the dots or the density of the dots. Therefore, a particular choice of a dot size necessitates a specific dot density to achieve the desired tone curve. The object is then to select the dot size, at each desired output density, that produces the "best" printed output. The dot density LUT 122 and dot size LUT 124 are designed by experimentally measuring the quality of the printed halftones at each combination of dot size and dot density, and then choosing the best combination of dot size and density for each desired output gray level. Because the dot density control and dot size control are designed by directly measuring the print quality, the resulting design accounts for attributes of both the printer and the halftoning system used.

Fig. 10 is a flow chart 180 showing a method of optimizing the dot size and dot density look-up tables using a closed loop measurement process in accordance with an embodiment of the present invention. As shown in Fig. 10, a test page is printed for each fixed dot size over the range of dot densities (block 182). Let n be the dot density, θ be the dot size, and g be the dot gray level. The dot gray level g is measured in units of gamma corrected absorbtance ranging from 0 to 1. So the normalized luminance is given by $l = (1 - g)^{\gamma}$ where $\gamma = 2.2$. For each dot density and dot size, the output absorptance, i.e., gray level $g_{\theta}(n)$, and the print distortion, $D_{\theta}(n)$ are measured from the printed test pages that are scanned at 600 dpi (block 184). Each test page consists of an array of 16 x 16 blocks printed with a fixed dot size and dot densities ranging from 0 to 255 where 255 corresponds to all dots on. The blocks are of size 0.35in. x 0.35in and placed in random order to avoid systematic measurement errors when printing across a page.

To measure the dot distortion $D_{\theta}(n)$ for each dot size θ and dot density n, a frequency weighted total squared error is used as the metric, where the frequency weighting is chosen to approximate the human visual system (HVS) response. The HVS model is a linear shift-invariant low pass filter. The frequency response of this filter is given by

$$H(u,v) = \frac{aL^b \exp(-\sqrt{u^2+v^2})}{c \ln L + d} \qquad \text{equ. 5}$$

where L is the average luminance in cd/m^2, a = 131.6, b = 0.3188, c = 0.525 and d = 3.91.

Let f(m, n) denote the measured gray level measured in units of gamma corrected absorptance so that I(m,n) = (1- f(m, n))$^\gamma$ where I(m,n) is the normalized absorptance and γ = 2.2. Then let

$$g = \frac{1}{N} \sum_{m,n} f(m,n) \qquad \text{equ. 6}$$

be the average absorptance level. The windowed error is computed as:

$$e(m,n) = (f(m,n) - g)w(m,n)(4) \qquad \text{equ. 7}$$

where w(m, n) is a Hamming window. The discrete Fourier transform (DFT) of e(m,n) is calculated to form E(k, *l*), and the print distortion metric is then given by

$$D = \sqrt{\frac{1}{MN} \sum_{m,n} |E(k,l)|^2 H(k,l)} \qquad \text{equ. 8}$$

where H(k, *l*) is the appropriately sampled version of H(u, v).

Figs. 11 and 12 are graphs 200 and 210 showing the measured tone curve, $g_\theta(n)$, i.e., the output absorptance, and the print distortion, $D_\theta(n)$, respectively, versus input dot density for a fixed value of dot size θ = 1, where a square dot shape is used as discussed in reference to Fig. 9. As shown in Figs. 11 and 12, the measurements of the tone curve and the distortion are noisy, so they must be appropriately smoothed. Smoothing the tone curve also ensures that the smoothed tone curve is monotonically increasing. Smoothing is performed using a combination of median and linear filtering and polynomial curve fitting, which are well known in the art. For example, the raw data of the tone curve may be smoothed with a median filter of length 3 that is applied 5 times followed by a linear filter that is 3 times with a filter kernel [0.25, 0.5, 0.25]. A 5th order polynomial curve fitting is then applied to obtain a smooth tone curve. The printing distortion curve, for example, may be smoothed by squaring the measured data, applying a median filter of length 9, applying a linear filter 40 times with a filter kernel [0.25, 0.5, 0.25], and then taking the square root of the result. The

tone-corrected ramp for each dot size is printed out to see if the resulting curve is reasonable. Of course, any desired method of smoothing may be used to generate a reasonably smooth curve.

Once the tone and distortion curves are determined for each dot size, e.g., θ = 0.3 to 1.4, these curves are used to determine the distortion at each gray level (block 186). Each tone curve is inverted to compute the value of the dot density required to produce each gray level.

$$n_\theta(\tilde{g}) = \max_n \{g_\theta(n) \le \tilde{g}\} \qquad \text{equ. 9}$$

Fig. 13 is a graph 220 showing a set of inverse tone curves for dot sizes ranging from 0.3 to 1.4. The x-axis is the desired output absorptance, while the y-axis is the dot density required to produce that absorptance level.

The inverse tone curves are used to determine the print distortion as a function of gray level.

$$D_{\theta,g} = D_\theta(n_\theta(g)) \qquad \text{equ. 10}$$

Fig. 14 is a graph 230 showing a set of plots of the print distortion versus the desired output absorptance for dot sizes ranging from 0.3 to 1.4, where each curve corresponds to $D_{\theta,g}$ as a function of g for fixed θ. These curves can be used to determine the best value of dot size for each desired output absorptance. For example, at the low values of absorptance, the smaller dot sizes generally produce lower distortion.

With these measurements, the optimized dot size LUT 124 and dot density LUT 122 can be calculated (blocks 188 and 190). These two look-up tables denoted by θ_i and n_i, respectively, where i ranges from 0 to 255. The look-up table functions, g_i and θ_i may be optimized so that both are smooth and achieve nearly optimal print quality at each desired absorptance level, by minimizing a cost function shown below as a function of the look-up tables θ_i.

$$\cos t = \sum_{i=0}^{255} D_{\theta_i,i}^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{255}(\theta_i - \theta_{i-1})^2 \qquad \text{equ. 11}$$

Here σ is a parameter that controls smoothness of the result by forcing the values of the parameters θ_i to only change in small amounts. Because the print distortion function is only measured for discrete values of the dot size θ, the function must be

interpolated using e.g., cubic spline interpolation, for missing values. Coordinate decent optimization is used to minimize the cost function; however, other optimization methods can be used and may be desirable, particularly ones that are robust to local minima in the cost functional.

Fig. 15 is a graph 240 showing a dot size curve for different σ values as a function of desired absorptance, where the optimized dot size LUT 124 curve 242, θ_i is $\sigma = 0.05$. The smooth curve 242 is used as the dot size LUT 124 for the AM/FM halftoning system 120, shown in Fig. 5. Smaller values of σ can be used to increase the smoothness of the dot size LUT, and larger values of σ can be used to reduce the smoothness of the LUT.

Once the look-up table θ_i is determined, the corresponding dot density LUT 122 n_i may be determined by using the inverse tone curves, $n_\theta(g)$ shown in graph 220 in Fig. 13, so that:

$$n_i = n_{\theta_i}(i) \qquad \text{equ. 12}$$

Again, because $n_\theta(i)$ is not measured for each value of dot size θ, the intermediate values may be interpolated using, e.g., cubic spline interpolation.

Fig. 16 is a graph 250 showing the dot density curve that corresponds to the curve 242 shown in graph 240 in Fig. 15 and is a function of the desired absorptance. The dot density curve shown in graph 250 is used as the dot density LUT 122 for the AM/FM halftoning system 120, shown in Fig. 5.

While graphs 240 and 250 may be used as the dot size LUT 124 and dot density LUT 122 in the AM/FM halftoning system 120, in practice, the best results may require some manual adjustment of these curves. For example, the dot density curve in graph 250 is not monotone. While the resulting gray levels of the AM/FM halftoning system 120 are still monotone due to the increasing dot size at those gray levels, in practice, it may be more desirable to constrain the solution to have monotone increasing dot density.

Figs. 17 and 18 are graphs 260 and 270, which show a dot size curve and dot density curve, respectively, as used with an eight-bit PWM code embodiment in which the pixel pairs are used, as described in reference to Figs. 7A and 7B.

It should be understood that variations of the present AM/FM halftoning system are possible. For example, Fig. 19 shows an embodiment of the AM/FM

halftoning system 300, which is similar to the embodiments shown in Fig. 3, like designated elements being the same. AM/FM halftoning system 300, however, includes a dot size diffusion unit 302, which performs error diffusion for dot size in a manner similar to the dot density diffusion system described above, e.g., in reference to Fig. 6. The dot size diffusion unit 302 operates with dot size modulation unit 110 within the AM modulation unit 304. This embodiment may be particularly useful when the PWM levels are limited. Thus, for example, if the PWM code is a two-bit code, four PWM levels are possible. With only four PWM levels, contouring artifacts or discontinuities will arise. With the use of dot size diffusion unit 302, the contouring artifacts may be minimized.

The source code in Appendices B1 and B2 written in C programming language are an example of an implementation of the present invention using a two-bit PWM code to control a printer system having a pulse width modulated laser capability for each printed pixel. Appendix B3 lists the particular coefficients used in the Tone-Dependent Error Diffusion process for the two-bit code used in conjunction with Appendices B1 and B2.

Fig. 20 shows a block diagram illustrating another possible embodiment of the present AM/FM halftoning system. Fig. 20 shows an integrated AM/FM halftoning system 350 in which the dot density control is integrated into the dispersed dot halftoning system 352 and the dot size control is integrated into the dot size modulation system 354. It should be understood, of course, that other variation exist, such as integrating only the dot density control into the dispersed dot halftoning system 352 or only the dot size control into the dot size modulation system 354. The integrated AM/FM halftoning system 350 has potential advantages of reducing quantization artifacts, reducing complexity, and improving halftone quality.

In one embodiment, the look-up tables and method for controlling the error diffusion process are provided on a computer readable medium, such as a microdiskette or floppy diskette as a printer driver. This printer driver is then installed into the computer so that the program is installed in the computer's RAM. Such a program may be also installed in the printer and, in one embodiment, installed in firmware within the printer. All logic functions may be implemented in hardware or software. Thus, for example, the image forming system may include a computer

that is coupled to the printing device, where a computer program executed by the computer includes instructions for implementing the functions of the present invention. The computer may be, e.g., a host computer, microprocessor or any other appropriate device. If hardware is used, the various table values would be available to the circuitry implementing the halftone method via bus lines. The method may also be carried out by an ASIC, which controls the timing and transfer of data to the various logic devices and look-up tables as well as to and from the image map, as would be understood by those skilled in the art after reading this disclosure.

While particular embodiments of the present invention have been shown and described, it will be obvious to those skilled in the art that changes and modifications may be made without departing from this invention in its broader aspects and, therefore, the appended claims are to encompass within their scope all such changes and modifications as fall within the true spirit and scope of this invention.

CLAIMS

What is claimed is:

1. A method of halftoning an image, said method comprising:

modulating the dot density of said image; and

modulating the dot size of printed dots to obtain a printed halftone image.

2. The method of Claim 1, further comprising:

inputting an input pixel value for a pixel location;

wherein said modulating the dot density modulates the spacing of the dot to be printed for said pixel location with respect to preceding and subsequent pixel locations and said modulating the dot size modulates the size of said dot to be printed for said pixel location.

3. The method of Claim 1, further comprising:

inputting an input pixel value for a pixel location within said image;

wherein modulating the dot density of said image comprises controlling the dot density for said pixel location within said image using said input pixel value and performing dispersed dot halftoning to produce a dot position based on said dot density.

4. The method of Claim 3, wherein said modulating the dot size of said image comprises controlling the dot size for said pixel location within an image using said input pixel value and performing dot size modulation based on said dot size and said dot position.

5. The method of Claim 4, wherein controlling the dot density for said pixel location comprises generating a dot density value based on said input pixel value, said dot density value being used to perform said dispersed dot halftoning, and wherein said controlling the dot size for said pixel location comprises generating a dot size value based on said input pixel value, said dot size value being used to perform dot size modulation.

6. The method of Claim 5, wherein said generating a dot density value is performed using a first look up table and generating a dot size value is performed using a second look up table.

7. The method of Claim 4, wherein controlling the dot density for said pixel location and performing dispersed dot halftoning are performed in a first integrated process and controlling the dot size for said pixel location and performing dot size modulation are also performed in a second integrated process.

8. The method of Claim 4, wherein said modulating the dot size of said image further comprises diffusing at least a portion of the dot size for the pixel location to at least one subsequently processed pixel location.

9. The method of Claim 1, wherein said modulating the dot density is performed using at least one of error diffusion, dispersed dot screening, and iterative search based halftoning.

10. The method of Claim 2, wherein said modulating the dot density is performed using tone dependent error diffusion.

11. The method of Claim 10, further comprising:

generating a dot density value based on said input pixel value, said dot density value being used in said tone dependent error diffusion;

said tone dependent error diffusion, comprising:

combining said dot density value with at least one previous error value to produce a modified pixel value;

comparing said modified pixel value with a threshold value to produce a halftone value for said pixel location; and

using said halftone value for said pixel location to produce an error value that is diffused to at least one subsequently processed pixel.

12. The method of Claim 1, wherein said modulating the dot size is performed using pulse width modulation.

13. The method of Claim 12, wherein said pulse width modulation provides a first value indicating the width of the pulse and a second value indicating the justification of said pulse within a pixel location.

14. The method of Claim 13, wherein when a printed dot is larger than one pixel, the pulse within adjacent pixels are justified together so that the dot may be formed with one continuous pulse.

15. A method of printing a halftone image, said method comprising:

inputting an input pixel value for a pixel location within an image to be printed;

generating a dot density control value using said input pixel value for said pixel location;

producing a halftone value for said pixel location using said dot density control value;

generating a dot size control value using said input pixel value for said pixel location; and

producing a size modulated halftone value for said pixel location based on said dot size control value and said halftone value.

16. The method of Claim 15, wherein said producing a halftone value for said pixel location also uses accumulated errors diffused from at least one other pixel location and provides an error for said pixel location to be diffused to at least one subsequently processed pixel location.

17. The method of Claim 15, wherein said producing a halftone value for said pixel location is performed using tone dependent error diffusion.

18. The method of Claim 15, wherein generating a dot density control value is performed using a first look up table and generating a dot size control value is performed using a second look up table.

19. The method of Claim 15, wherein said producing a size modulated halftone value is performed using pulse width modulation.

20. The method of Claim 19, wherein said pulse width modulation is performed using a look up table.

21. The method of Claim 15, further comprising diffusing at least a portion of said dot size value for the pixel location to at least one subsequently processed pixel location.

22. A method of optimizing a dot size look-up table and a dot density look-up table for a printing system that uses dot size modulation and dot density modulation, the method comprising:

printing at least one test page showing the combinations of dot sizes and dot densities;

measuring the output absorptance for each combination and the print distortion for each combination;

determining the print distortion at each output absorptance using the measured output absorptance and the measured print distortion for each combination;

calculating the optimized dot size look-up table using the print distortion at each output absorptance; and

calculating the optimized dot density look-up table using the print distortion at each output absorptance.

23. The method of Claim 22, wherein determining the print distortion comprises:

inverting the output absorptance for each combination to compute the value of the dot density required to produce each output absorptance; and

using the inverted output absorptance and the print distortion for each combination to determine the print distortion as a function of output absorptance.

24. The method of Claim 22, wherein calculating the optimized dot size look-up table comprises minimizing a cost function of the print distortion at each output absorptance as a function of dot size.

25. The method of Claim 24, wherein calculating the optimized dot density look-up table further comprises using the calculated optimized dot size look-up table.

26. A printing system including a printer and a computer, the printing system printing a dot density and dot size modulated image, the method performed by the printing system comprising:

performing dispersed dot halftoning for a pixel location based on an input pixel value for said pixel location;

performing dot size modulation for said pixel location based on the results of said dispersed dot halftoning and said input pixel value.

27. The image forming device of Claim 26, wherein the printer is one of an electrophotographic printing device, electrophotographic copying device, and an inkjet printer.

28. An image forming system comprising:

a computer;

a printing device coupled to said computer;

a computer program executed by said computer, wherein said computer program comprises computer instructions for:

receiving an image;

modulating the dot density of said image; and

modulating the dot size of printed dots to obtain a halftone for said image.

29. The image forming system of Claim 28, wherein:

receiving an image comprises receiving an input pixel value for a pixel location within said image;

modulating the dot density of said image comprises generating a dot density control value using said input pixel value for said pixel location and producing a halftone value for said pixel location using said dot density control value; and

modulating the dot size comprises generating a dot size control value using said input pixel value for said pixel location and producing a size

modulated halftone value for said pixel location based on said dot size control and said halftone value.

30. The image forming system of Claim 29, wherein said producing a halftone value for said pixel location comprises adding at least a portion of accumulated errors from at least one other pixel location with said dot density control value and diffusing an error for said pixel location to at least one subsequently processed pixel location.

31. The image forming system of Claim 28, wherein said computer is a microprocessor.

32. The image forming system of Claim 28, wherein said image forming system is one of an electrophotographic printing device, electrophotographic copying device, and an inkjet printer.

33. An image forming device comprising:

a dot density control unit;

a dot size control unit;

a dispersed dot halftoning unit coupled to said dot density control unit; and

a dot size modulation unit coupled to said dispersed dot halfoning unit and said dot size control unit; and

a printing device coupled to said dot size modulation unit.

34. The image forming device of Claim 33, wherein:

said dot density control unit is configured to receive an image and produce a dot density value for at least a portion of said image;

said dot size control unit is configured to receive said image and produce a dot size value for said at least a portion of said image;

said dispersed dot halftoning unit is configured to receive said dot density value and produce a halftone value for said at least a portion of said image;

said dot size modulation unit is configured to receive said halftone value and said dot size value and produces an output signal representing said

halftone value and a modulated size value for said at least a portion of said image;

said printing device is configured to receive said output signal and print said at least a portion of said image.

35. The image forming device of Claim 33, wherein said dispersed dot halftoning unit is a tone dependent error diffusion unit.

36. The image forming device of Claim 33, further comprising a dot size diffusion unit coupled to said dot size modulation unit.

37. The image forming device of Claim 33, wherein said dot density control unit is integrated into said dispersed dot halftoning unit.

38. The image forming device of Claim 33, wherein said dot size control unit is integrated into said dot size modulation unit.

ABSTRACT

A combined dot density and size modulation system uses dispersed dot halftoning in conjunction with dot size modulation to produce a halftone image in which both the density and size of the dots are modulated to control overall gray level. The dot density and size modulation system offers advantages over pure dot density modulation systems or pure dot size modulation systems because it allows an extra degree of flexibility which can be used to increase the visual quality of the halftoned pattern and/or increase the robustness of the halftoning to printer artifacts and variations. An input pixel value is used to independently produce a dot density value and a dot size value. The dot density value and dot size values may be obtained from, e.g., look up tables that have been optimized for print quality and printer stability. Dispersed dot halftoning is used to provide a halftone value for the desired pixel location using the dot density value. The dispersed dot halftoning may be, e.g., tone dependent error diffusion. The halftone value and the dot size value for the pixel location is then used to generate a modulated code, e.g., a pulse width modulated code, to the printer. The modulated code may include both the pulse width of the desired dot for the pixel location as well as the justification, e.g. left, center, or right, for the pixel location. The dot density and size modulation system is particularly useful in modern electrophotographic printing systems that allow the printed dot size to be almost continuously varied through the specification of a pulse width modulation (PWM) code.

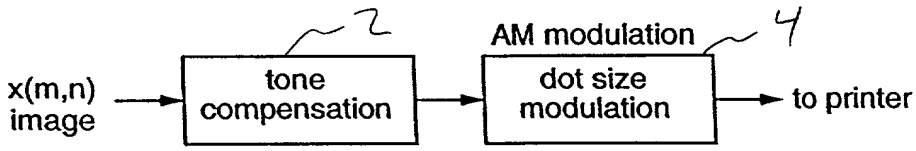


Fig 1

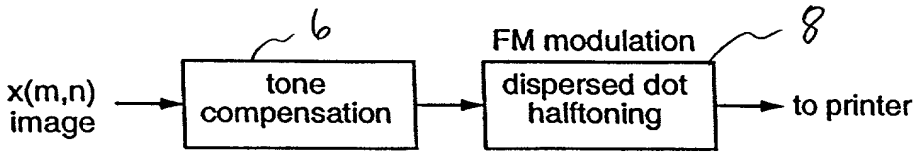


Fig 2

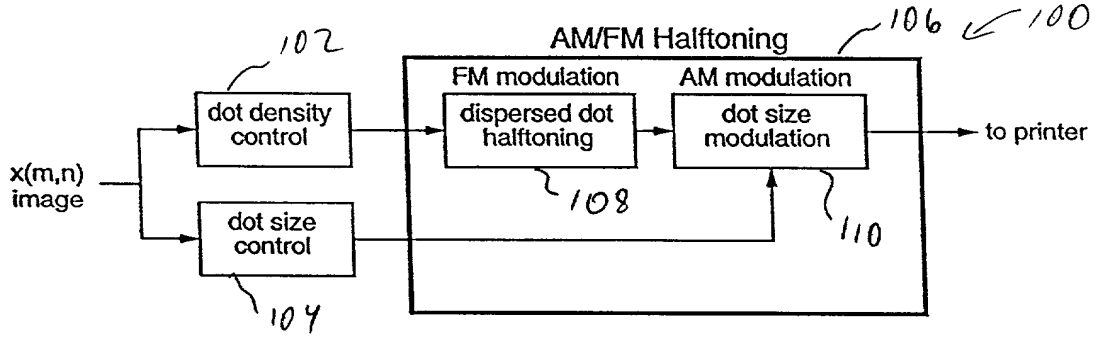


Fig 3

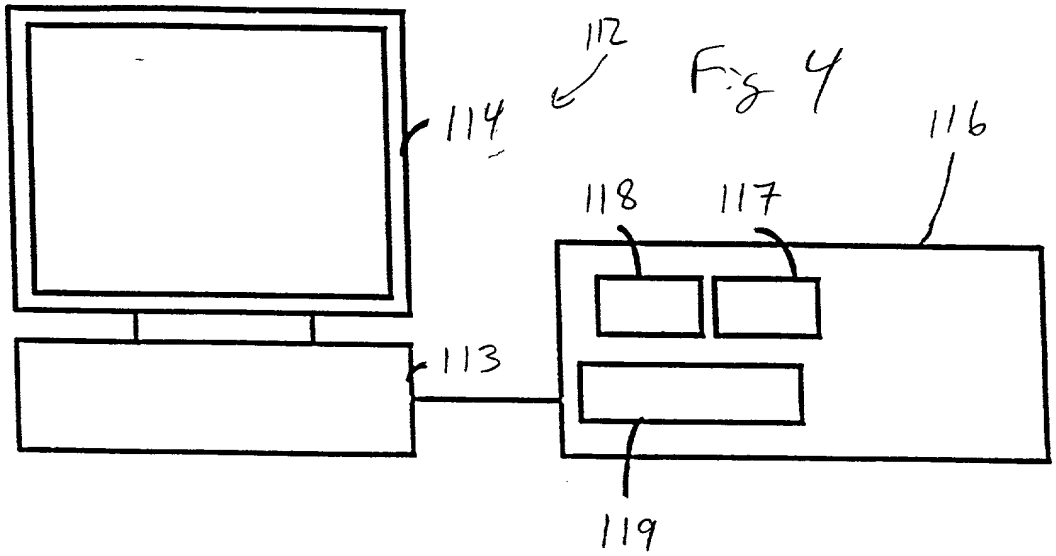


Fig 4

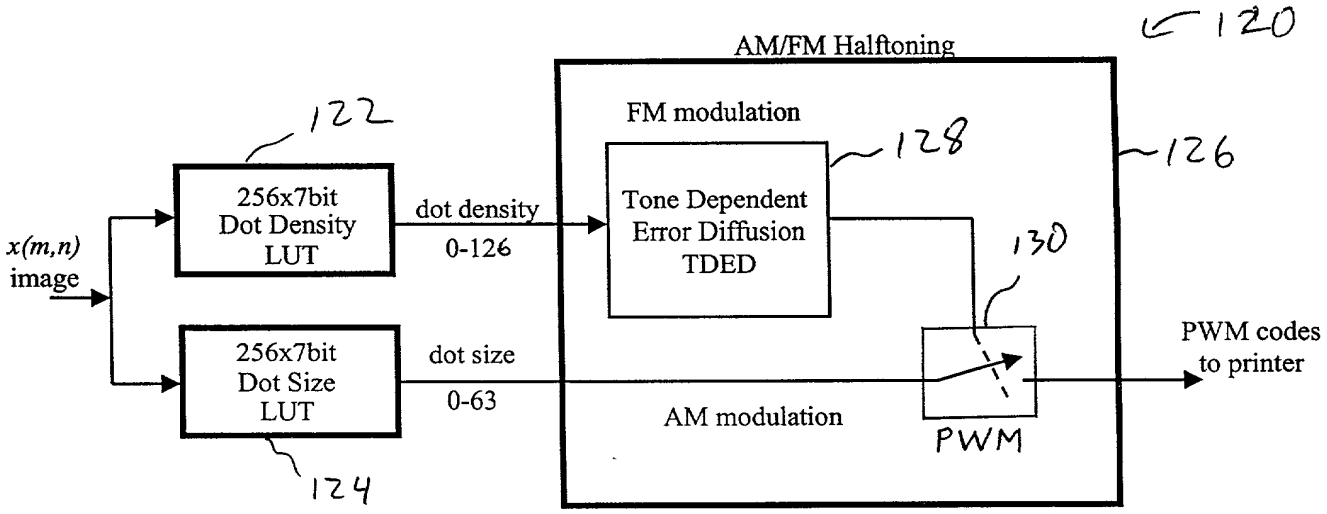

120
AM/FM Halftoning
FM modulation
122
128
126
256x7bit
Dot Density
LUT
dot density
0-126
Tone Dependent
Error Diffusion
TDED
130
x(m,n)
image
256x7bit
Dot Size
LUT
dot size
0-63
PWM codes
to printer
AM modulation
PWM
124

Fig 5

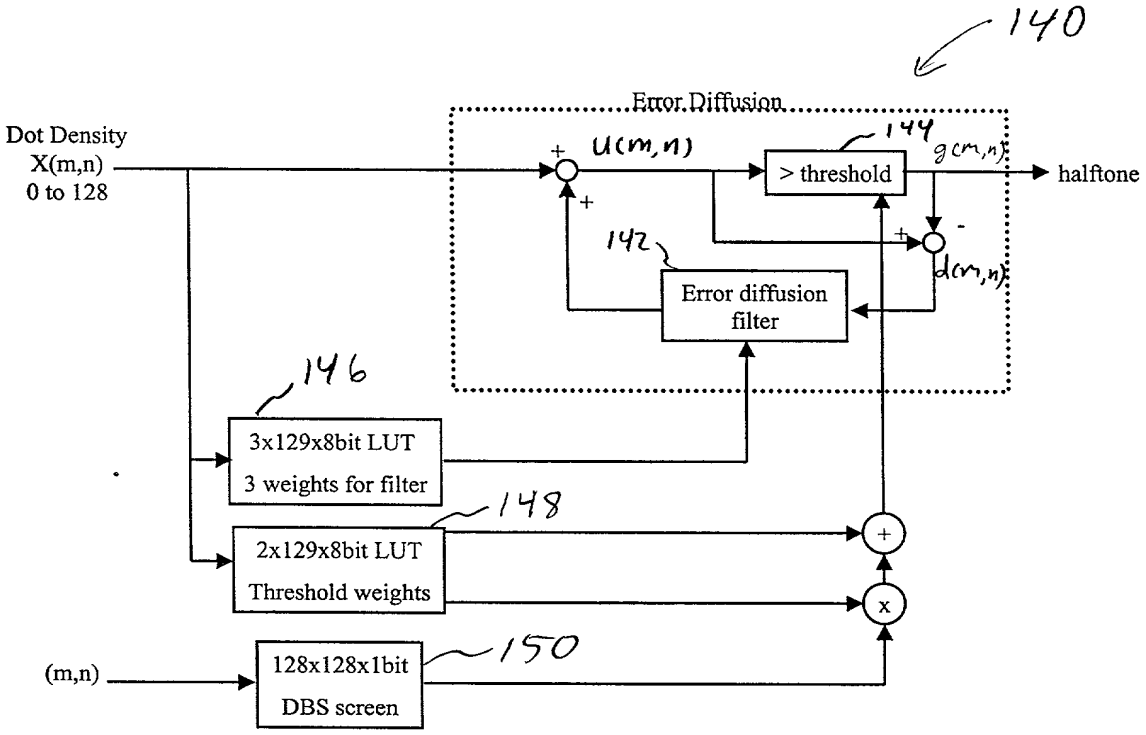

140
Error Diffusion
Dot Density
X(m,n)
0 to 128
U(m,n)
144
> threshold
g(m,n)
halftone
142
d(m,n)
Error diffusion
filter
146
3x129x8bit LUT
3 weights for filter
148
2x129x8bit LUT
Threshold weights
(m,n)
128x128x1bit
DBS screen
150

Fig 6

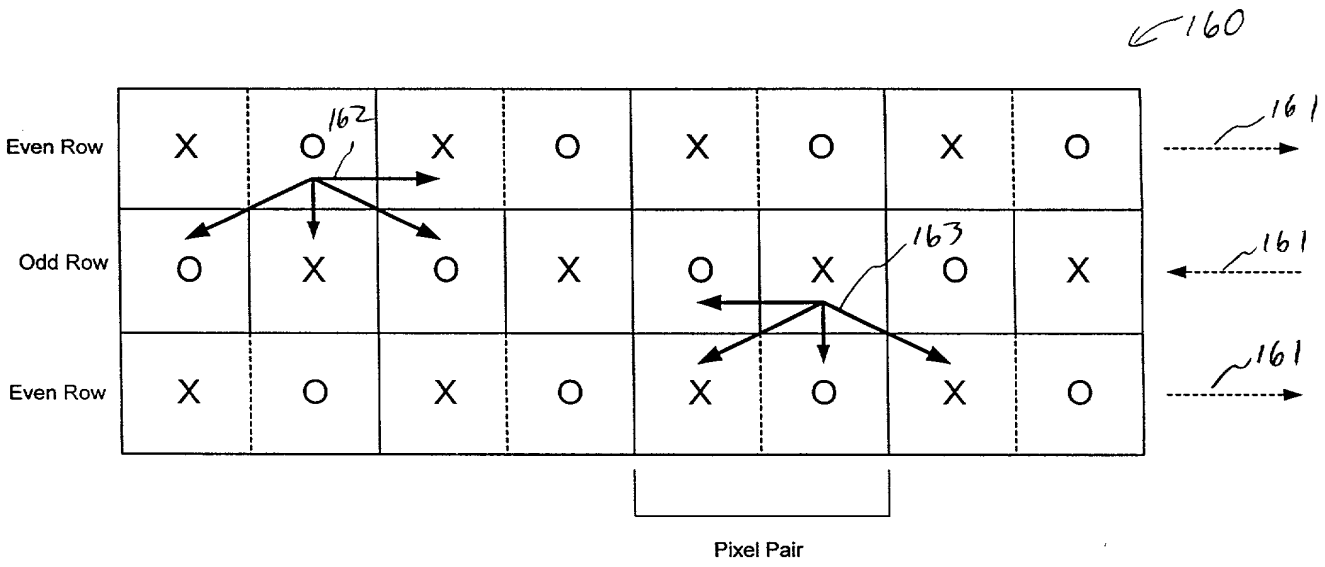


Fig 7A

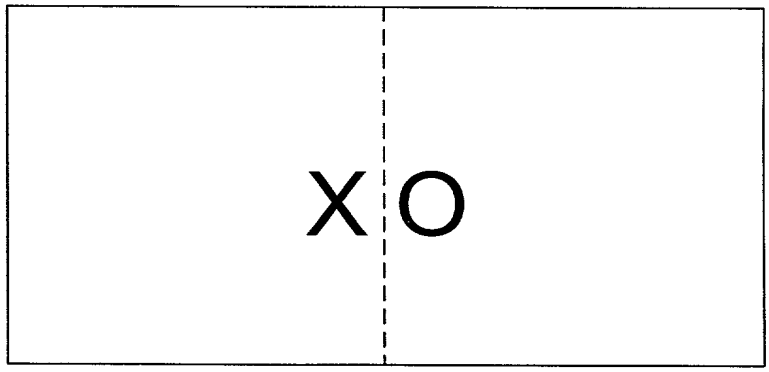


Fig 7B

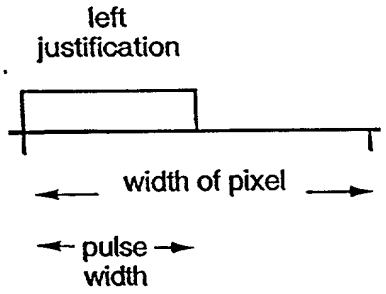


Fig 8A

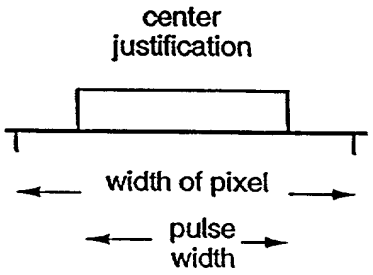


Fig 8B

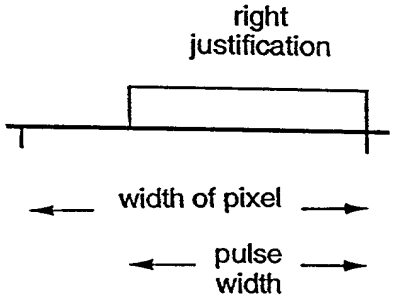


Fig 8C

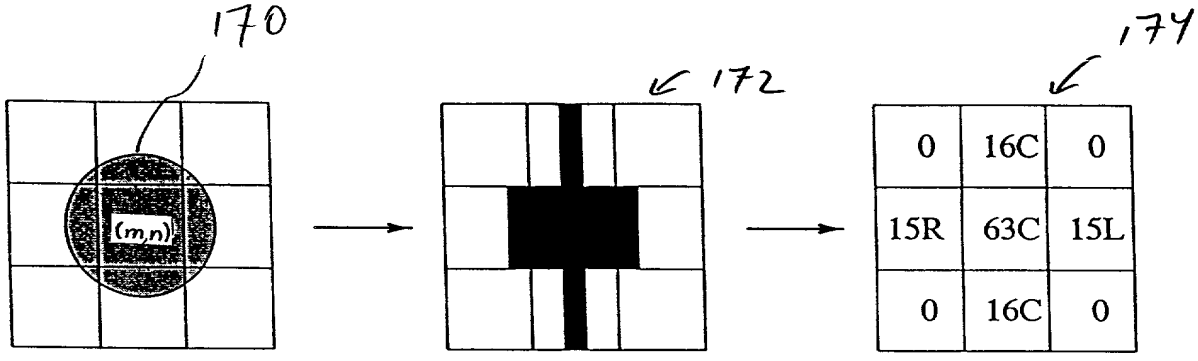


Fig 9

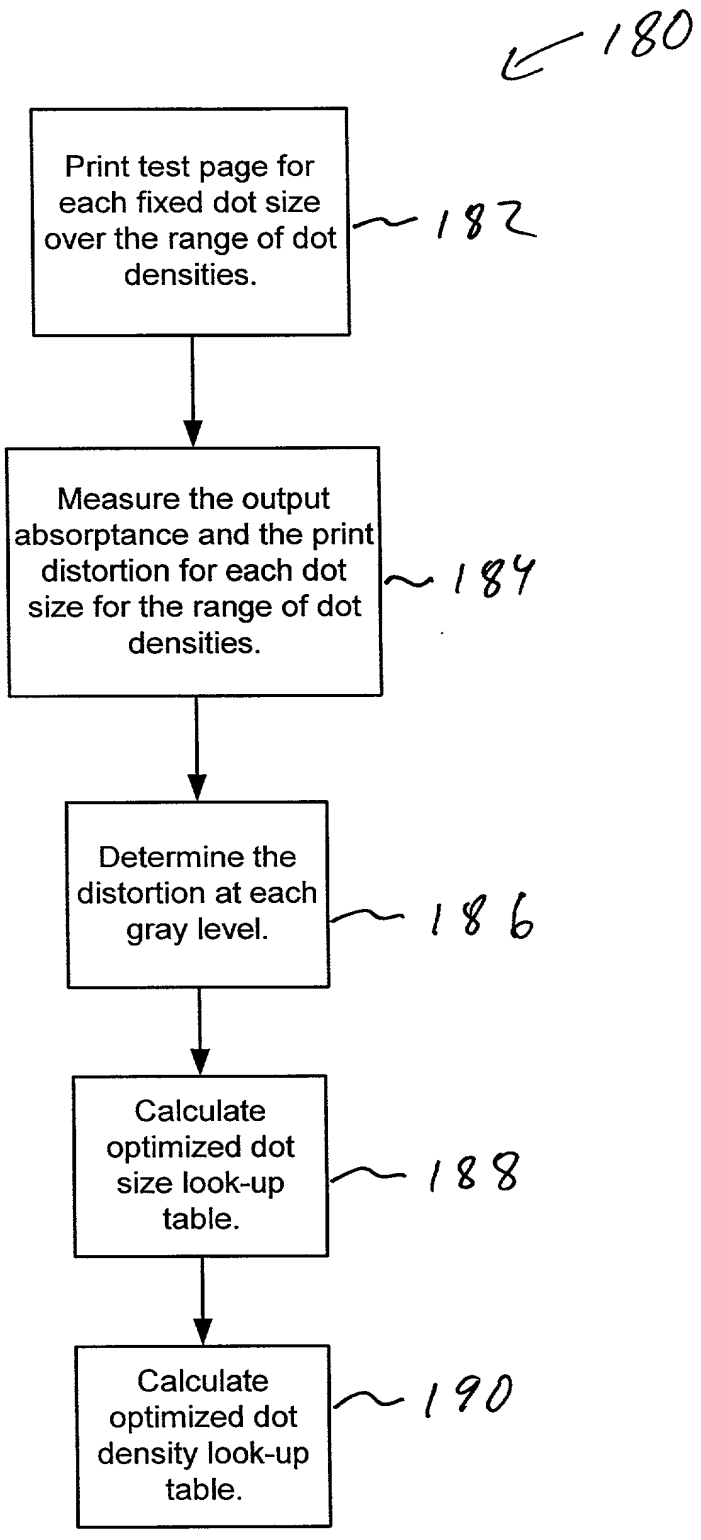
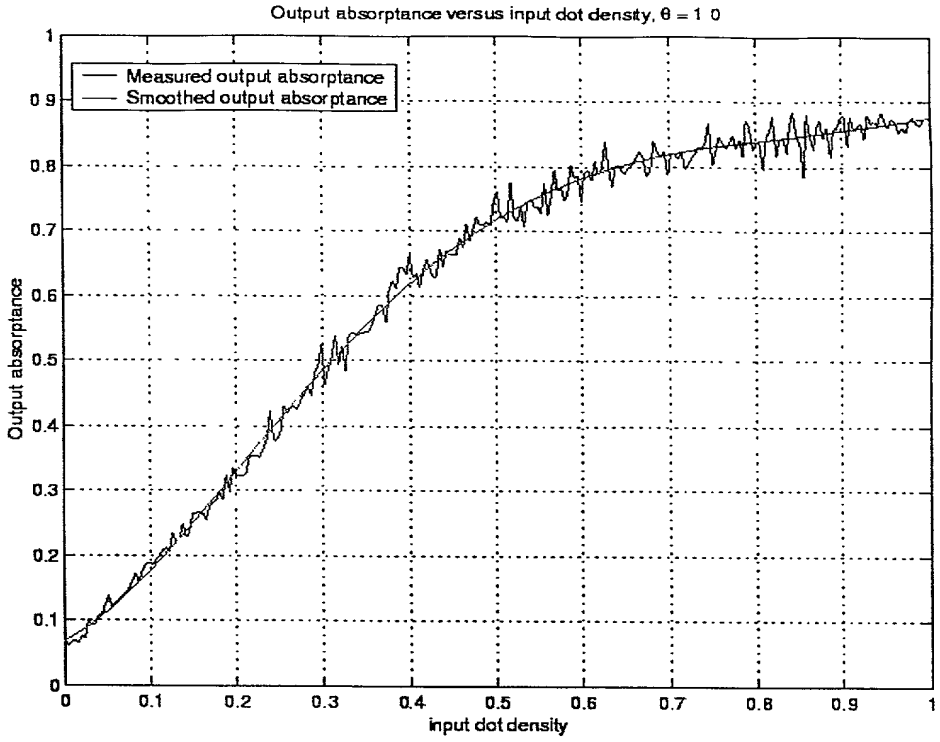
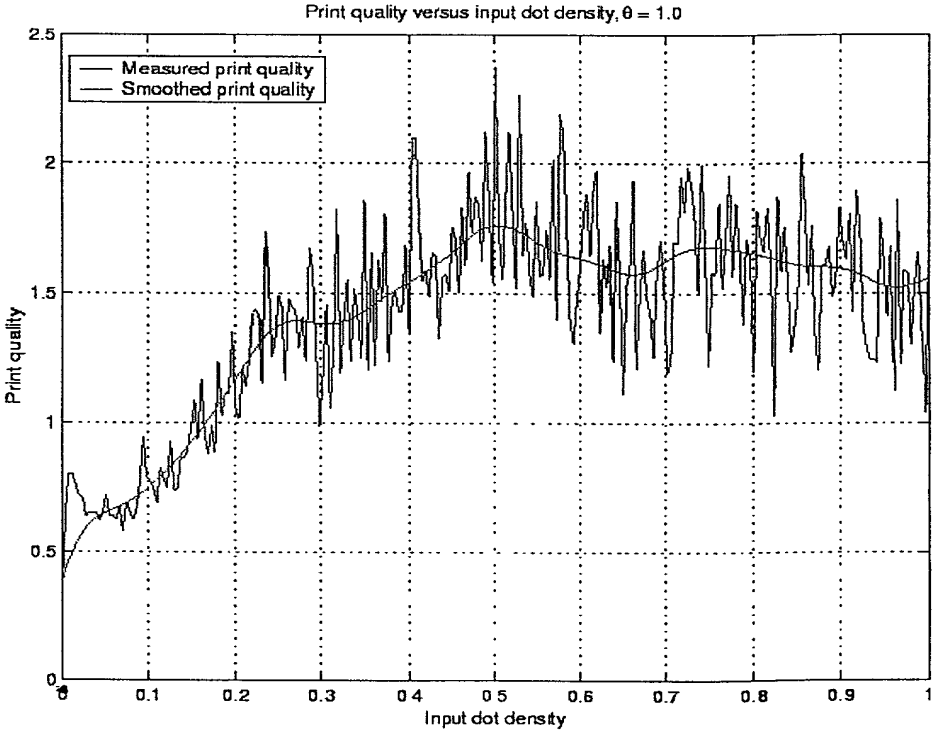


Fig 10



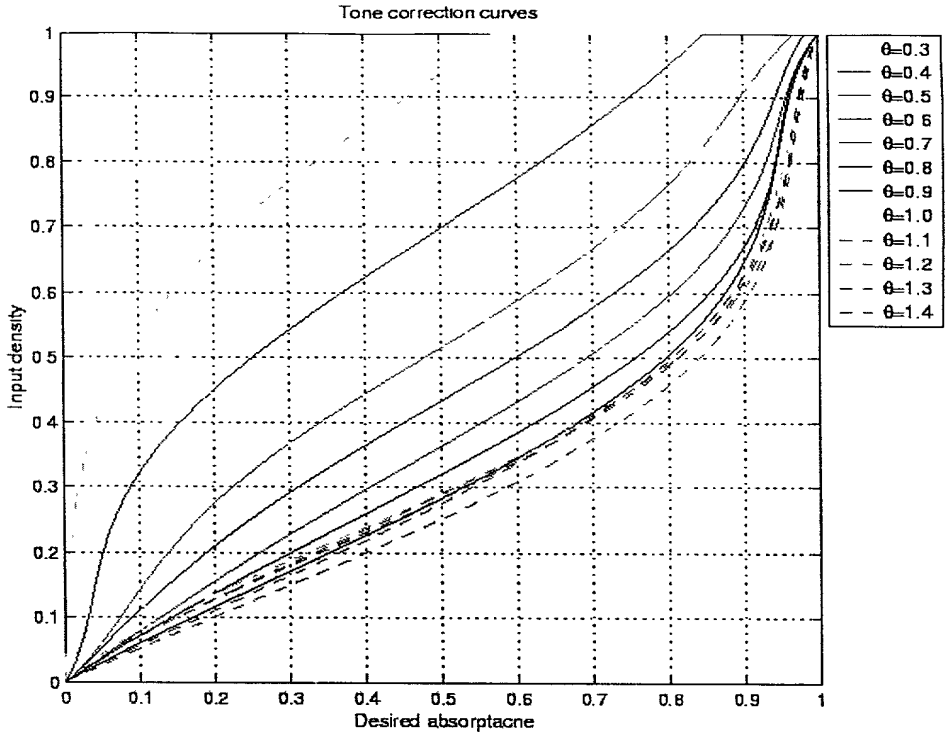
200

Fig 11



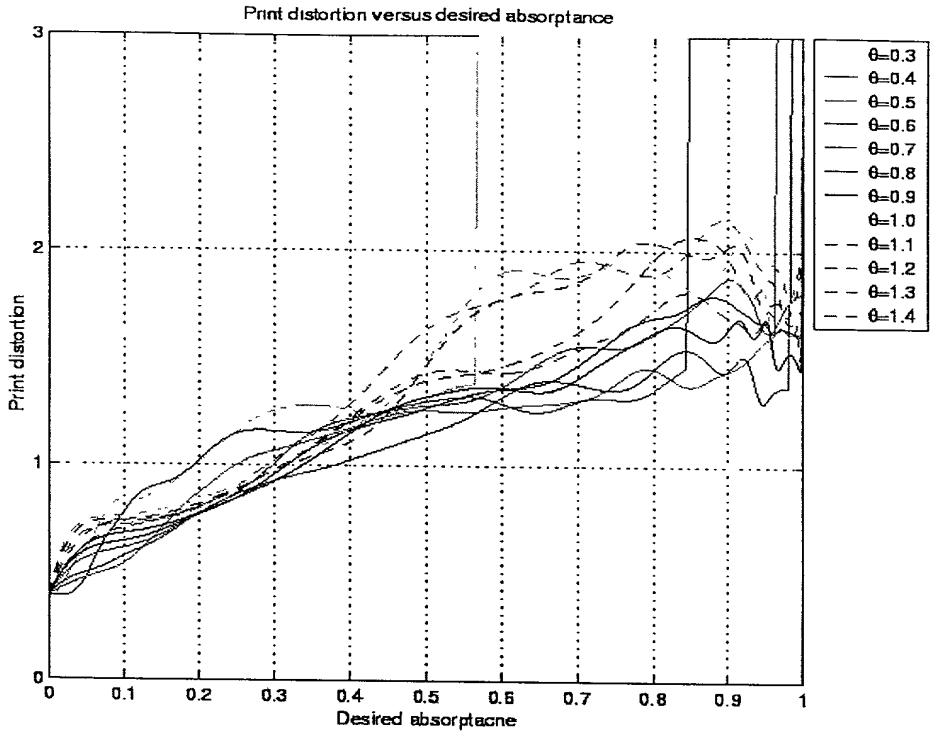
210

Fig 12



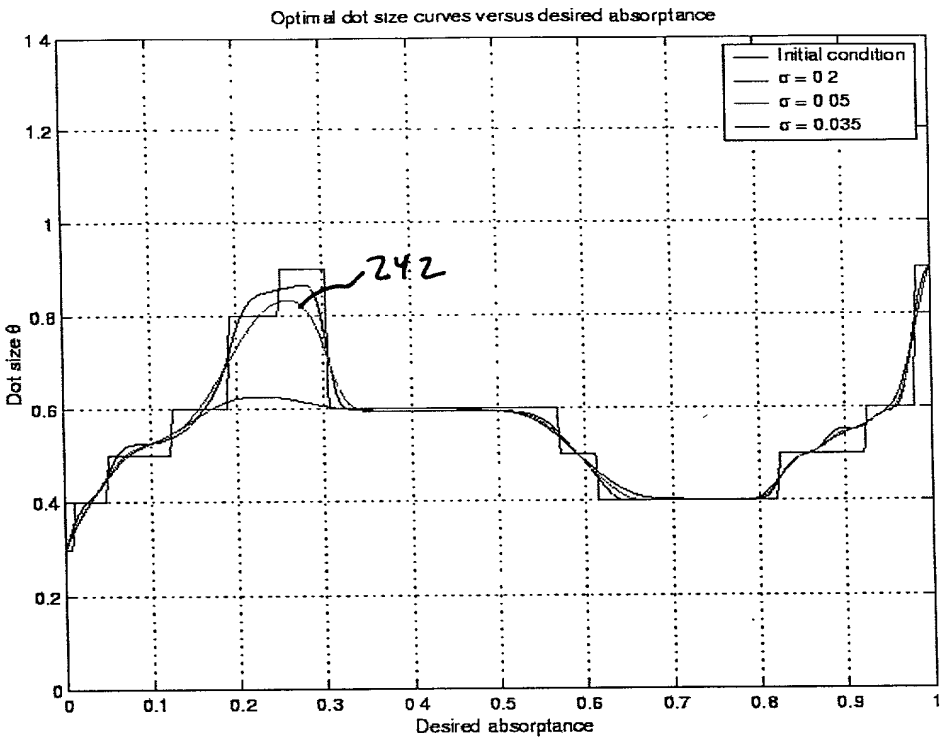
220

Fig 13



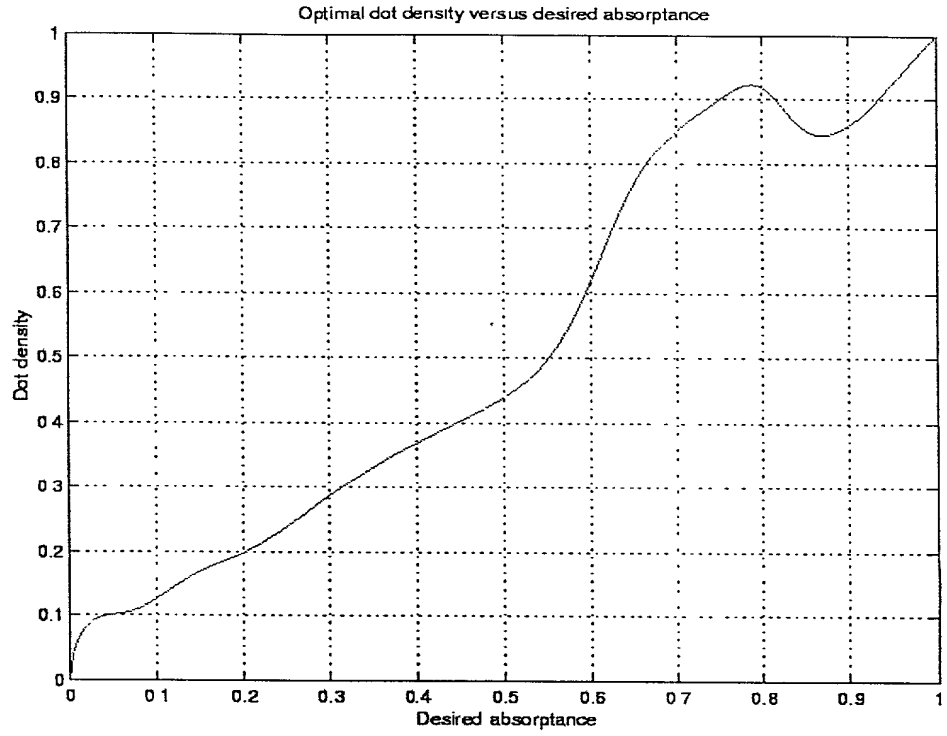
230

Fig 14



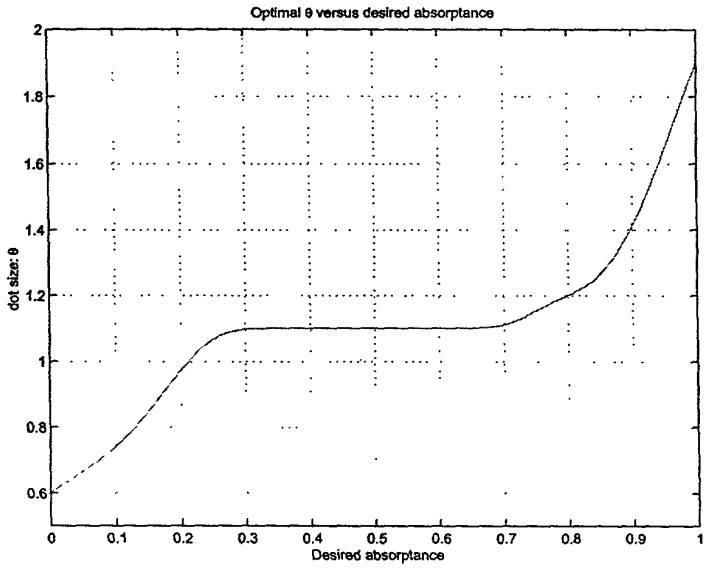
240

Fig 15



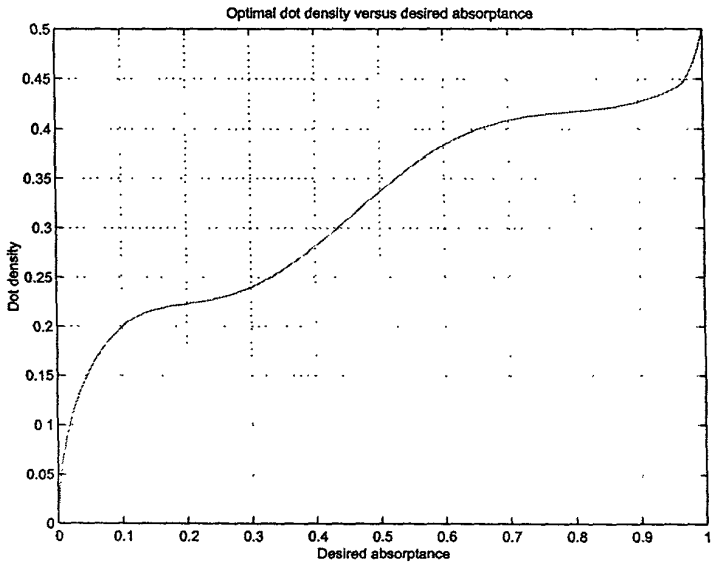
250

Fig 16



260

Fig 17



270

Fig 18

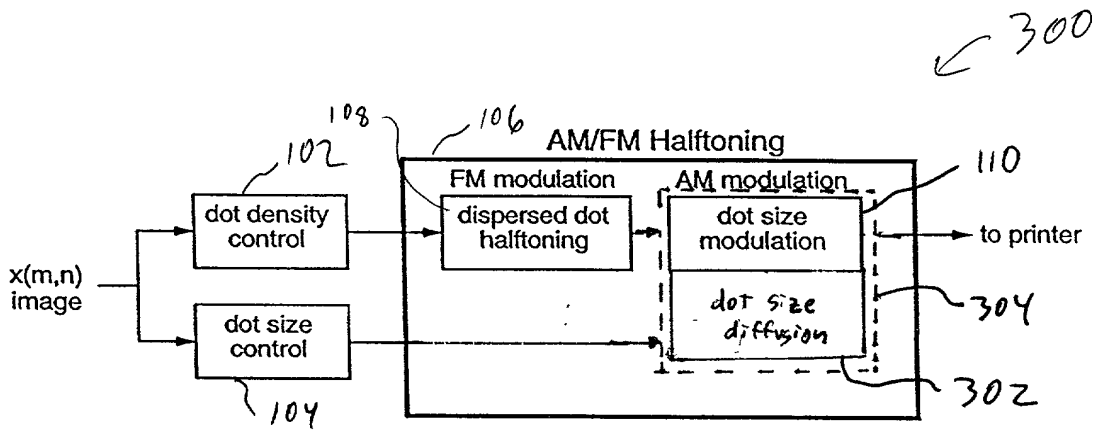

300
108
106
AM/FM Halftoning
102
FM modulation
AM modulation
110
dot density control
dispersed dot halftoning
dot size modulation
to printer
x(m,n) image
dot size control
dot size diffusion
304
302
104

Fig 19

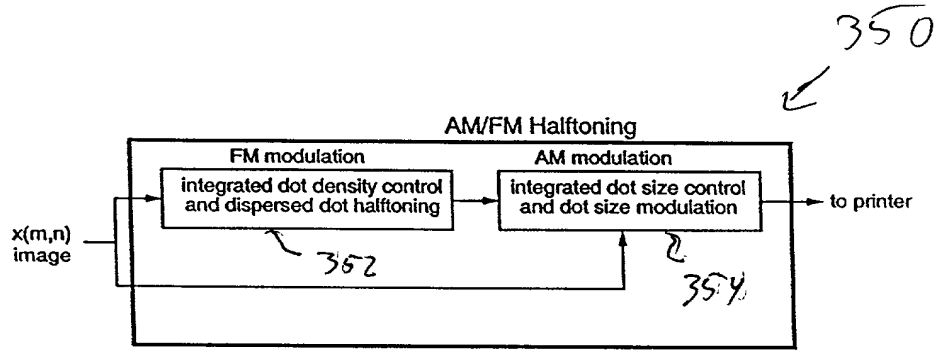

350
AM/FM Halftoning
FM modulation
AM modulation
integrated dot density control and dispersed dot halftoning
integrated dot size control and dot size modulation
to printer
x(m,n) image
352
354

Fig 20

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

ATTORNEY DOCKET NO. 10003284-1

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

Combined Dot Density And Dot Size Modulation

the specification of which is attached hereto unless the following box is checked:

( ) was filed on _____ as US Application Serial No. or PCT International Application Number _____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
			YES: ___ NO: ___
			YES: ___ NO: ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Customer Number 022879

Place Customer Number Bar Code Label here

Send Correspondence to:
HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P.O. Box 272400
Fort Collins, Colorado 80527-2400

Direct Telephone Calls To:

Lane R. Simmons

(208) 396-3633

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Zhen He **Citizenship:** CN

Residence: 843 Hartford Street, Apt. 6, West Lafayette, IN 47904

Post Office Address: Same as Residence

Inventor's Signature **Date**

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION (continued)

ATTORNEY DOCKET NO. 10003284-1

Full Name of # 2 joint inventor: Charles A. Bouman Citizenship: US

Residence: 40 Clay Ct., West Lafayette, IN 47906

Post Office Address: Same as residence

Inventor's Signature _____ Date _____

Full Name of # 3 joint inventor: Qian Lin Citizenship: US

Residence: 159 Gilbert Ave., Santa Clara, CA 95051

Post Office Address: 1501 Page Mill Road, Palo Alto. CA 94304

Inventor's Signature _____ Date _____

Full Name of # 4 joint inventor: _____ Citizenship: _____

Residence: _____

Post Office Address: _____

Inventor's Signature _____ Date _____

Full Name of # 5 joint inventor: _____ Citizenship: _____

Residence: _____

Post Office Address: _____

Inventor's Signature _____ Date _____

Full Name of # 6 joint inventor: _____ Citizenship: _____

Residence: _____

Post Office Address: _____

Inventor's Signature _____ Date _____

Full Name of # 7 joint inventor: _____ Citizenship: _____

Residence: _____

Post Office Address: _____

Inventor's Signature _____ Date _____

Full Name of # 8 joint inventor: _____ Citizenship: _____

Residence: _____

Post Office Address: _____

Inventor's Signature _____ Date _____

Appendix A1

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* amfm_8bit.c file */

/* 8 bits/pixel am/fm halftoning algorithm (with partial doting) */
/* Image length and width are assumed to be multiple of 8 */
/* One row serpentine TDED with suppressing each other dot */
/* The output is a tiff file containing 8bit pwm codes */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "coef.h"
#include "tiff.h"
#include "allocate.h"

void amfm(unsigned int,unsigned int,unsigned char *,unsigned char *,\
        short *,unsigned char **,TDEDPARA *,short *,short *);
void amfm_ed_8bits(unsigned int,unsigned int,unsigned char **,unsigned char**,\
                unsigned char **,TDEDPARA *,short *,short *);
int main(int argc, char ** argv)
{
  int i,j;
  unsigned int height,width;
  FILE * fp;
  struct TIFF_img input_img, output_img, mid;
  time_t first, second;
  TDEDPARA *tdedpara = &TDEDcoeff[0];
  short *dotdensityLUT = &OptDensityLUT[0];
  short *dotsizeLUT = &OptSizeLUT[0];


  if(argc<3) {
      printf("usage: %s input_img.tif output_img.tif\n",argv[0]);
      return 1;
  }

  /* read the input image  */
  if ((fp=fopen(argv[1], "rb"))==NULL) {
```

```
    printf("can not open file %s 1\n",argv[1]);
    exit(2);
  }
  if(read_TIFF(fp,&input_img)) {
    printf("error reading input file\n");
    exit(3);
  }
  fclose(fp);

  if((fp=fopen("dbshalf.tif", "rb"))==NULL) {
    fprintf(stderr, "can not open file: dbshalf.tif\n");
    exit(1);
  }
  if(read_TIFF(fp,&mid))
  {
    fprintf(stderr, "error reading file\n");
    exit(1);
  }
  fclose(fp);

  /* Set variable to do timing of algorithm */
  first = time(NULL);

  /* Modify image width to make sure each strip is multiple of 8 */
  width = floor(input_img.width/8)*8;
  height = floor(input_img.height/8)*8;

  /* Allocate memory for entire fm output image. */
  get_TIFF( &output_img, height, width, 'g' );

  amfm_ed_8bits(height,width,input_img.mono,output_img.mono,\
                mid.mono,tdedpara,dotdensityLUT,dotsizeLUT);

  /* show the run time */
  second = time(NULL);
  fprintf(stdout,"\nFinished AM/FM and writing results.\n");
  fprintf(stdout,"Cum. run time: %f sec.\n",difftime(second,first));

  /* write PWM codes image */
  if( (fp = fopen(argv[2],"wb"))==NULL) {
    printf ("cannot open file %s\n", argv[2]);
    exit(4);
  }

  if(write_TIFF(fp,&output_img)) {
    printf ("\nError writing TIFF file %s\n", argv[2]);
    return 1;
  }
  fclose(fp);

  /* free the space */
  free_TIFF(&(output_img));
  free_TIFF(&(input_img));
  free_TIFF(&(mid));
  fflush(stdout);
  return 0;
}
```

```
void amfm_ed_8bits(
  unsigned int height,         /* Input image height */
  unsigned int width,          /* Input image width */
  unsigned char ** contone_img,      /* Input image [height][width] */
  unsigned char ** token_img, /* Output token image [height][width] */
  unsigned char ** dbs_screen,       /* DBS screen used in thresholding of fm
part */
  TDEDPARA *tdedpara,          /* Tone-dependent error diffusion parameters */
  short *dotdensityLUT,        /* Optimal dot density curve */
  short *dotsizeLUT)           /* Optimal dot size curve */
{
  short *fm_err;
  unsigned int i,j;

  /* initialize first row of fm error buffer */
  srand(1); /* fix the seed */
  fm_err = (short*)malloc(sizeof(short) * (width+2));
  for(j = 0; j<width+2; j++)
    fm_err[j] = (rand()%128-64);  /* initialization */

  /* Process the input image with 2 rows each time */
  for(i=0; i<height; i+=2) {
    if((i%600) == 0) printf("amfm_ed: starting row %d\n", i);
    amfm(width,i,contone_img[i],token_img[i],fm_err,dbs_screen,\
         tdedpara,dotdensityLUT,dotsizeLUT);
  }

  free(fm_err);
  return;

}

/* This subroutine only processes 2 rows */
/* Assume width of image is multiple of 8 */
void amfm(
  unsigned int width,   /* Input image width */
  unsigned int i,       /* ith row */
  unsigned char *img_in,      /* ith row of input image array */
  unsigned char *img_out,     /* ith row of output image array */
  short *fm_err,        /* FM error buffer */
  unsigned char ** dbs_screen,      /* dbs_screen[SCREENHEIGHT][SCREENWIDTH] */
  TDEDPARA *tdedpara,   /* Tone-dependent error diffusion parameters */
  short *dotdensityLUT, /* Optimal dot density curve */
  short *dotsizeLUT)    /* Optimal dot size curve  */

{
  short fm_tmp,thresholding;
  short *fm_err_ptr,*tded_ptr;
  short pixela, pixelb, output;
  int j;
  unsigned char *img_in_ptr, *img_out_ptr, *dbs_pat_rowptr;
  short dotdensity, mod_input, error;
  short W1, W2, W3, W4, T2, DT, e1, e2, e3, e4;
  FILE *fp;

  /*-------------------------------------*/
```

```
/* serpentine even rows                */
/*-------------------------------------*/
    /* initial points */
fm_tmp = 0;
fm_err_ptr = fm_err+1;
img_in_ptr = img_in;
img_out_ptr = img_out;

dbs_pat_rowptr = dbs_screen[(i++)%SCREENHEIGHT]; /* Inverse dbs pattern */

/* Index through pixels in pairs */
for(j = 0; j<width; j=j+2 ) {

  /* First process FM (dot density) for left pixel in pixel pair. */

  /* Get first pixel */
  pixela = *(img_in_ptr++);

  /* Use look-up-table to get dot density */
  dotdensity = dotdensityLUT[pixela];

  /* Compute look-up table entries for tone dependent error diffusion */
  tded_ptr = (short*)(tdedpara + dotdensity);
  T2 = *(tded_ptr++);
  DT = *(tded_ptr++);
  W1 = *(tded_ptr++);
  W2 = *(tded_ptr++);
  W3 = *(tded_ptr++);
  W4 = *tded_ptr;

  /* compute dotdensity modified by diffused error */
  mod_input = dotdensity + *fm_err_ptr;

  /* Threshold modifed dotdensity */
  thresholding = mod_input - (dbs_pat_rowptr[j%SCREENWIDTH] * DT + T2);
  output = (thresholding > 0) ? 255 : 0;

  /* Compute weighted errors */
  error = output - mod_input;
  e1 = (W1 * error)>>8;
  e2 = (W2 * error)>>8;
  e3 = (W3 * error)>>8;
  /*e4 = (W4 * error)>>8;*/
  e4 = error - e1 - e2 -e3;
  /* Diffuse error forward in 1-D error buffer */
  *(--fm_err_ptr) -= e4;
  *(++fm_err_ptr) = fm_tmp - e3;
  *(++fm_err_ptr) -= e1;
  fm_tmp = -e2;

  /* Now process FM (dot density) for right pixel in pixel pair. */
  /* Use same TDED parameters as for left pixel. */

  /* Get second pixel */
  pixelb = *(img_in_ptr++);

  /* Use look-up-table to get dot density */
```

```
    dotdensity = dotdensityLUT[pixelb];

    mod_input = dotdensity + *fm_err_ptr;
    error = - mod_input;  /* suppress dot firing at this pixel */

    e1 = (W1 * error)>>8;
    e2 = (W2 * error)>>8;
    e3 = (W3 * error)>>8;
    /*e4 = (W4 * error)>>8;*/
    e4 = error - e1 - e2 -e3;
    /* Using the tded weights of the left pixel */
    *(--fm_err_ptr) -= e4;
    *(++fm_err_ptr) = fm_tmp - e3;
    *(++fm_err_ptr) -= e1;
    fm_tmp = -e2;

/* Begin section on dot size rendering with partial doting */
    if(output) {
      /* Left pixel */
      *(img_out_ptr++) = (dotsizeLUT[pixela]>>1)+NEWRIGHT;
      /* Right pixel */
      if(dotsizeLUT[pixela] & 1)    /* Take care of quantization error */
         *(img_out_ptr++) = ((dotsizeLUT[pixelb]+1)>>1) + NEWLEFT;
      else
         *(img_out_ptr++) = (dotsizeLUT[pixelb]>>1) + NEWLEFT;
    }
    else {
      *(img_out_ptr++) = NEWRIGHT;
      *(img_out_ptr++) = NEWLEFT;
    }
  }   /* end of ith row */


/*-------------------------------------*/
/* serpentine odd rows                 */
/*-------------------------------------*/
  fm_tmp = 0;
  /* Set fm error buffer pointer to the end of fm_err buffer */
  fm_err_ptr = fm_err+ width - 1;  /* offset by 1 */
  img_in_ptr = img_in+width*2-2;
  img_out_ptr = img_out+width*2-1;
  *img_out_ptr = NEWRIGHT;  /* Take care of the last pixel in odd row */
  img_out_ptr -= 2;

  dbs_pat_rowptr = dbs_screen[i%SCREENHEIGHT];

  /* Index through pixels in pairs */
  for(j = width-2; j> 0; j=j-2) {
  /* First process FM (dot density) for right pixel in pixel pair */

  /* Get right pixel */
    pixela = *(img_in_ptr--);

    /* Use look-up-table to get dot density */
    dotdensity = dotdensityLUT[pixela];

    /* Compute look-up table entries for tone dependent error diffusion */
```

```
    tded_ptr = (short*)(tdedpara + dotdensity);

    T2 = *(tded_ptr++);
    DT = *(tded_ptr++);
    W1 = *(tded_ptr++);
    W2 = *(tded_ptr++);
    W3 = *(tded_ptr++);
    W4 = *tded_ptr;

    /* Compute dotdensity modified by diffused error */
    mod_input = dotdensity + *fm_err_ptr;

    /* suppress this dot and compute the error */
    error = - mod_input;

    /* Compute weighted errors */
    e1 = (W1 * error)>>8;
    e2 = (W2 * error)>>8;
    e3 = (W3 * error)>>8;
    /*e4 = ((W4 * error)>>8);*/
    e4 = error - e1 - e2 -e3;

    /* duffuse error forward in 1-D error buffer */
    *(++fm_err_ptr) -= e4;
    *(--fm_err_ptr) = fm_tmp - e3;
    *(--fm_err_ptr) -= e1;
    fm_tmp = -e2;

    /* Now process FM (dot density) for Left pixel in a pair */

    /* Get second pixel */
    pixelb = *(img_in_ptr--);

    /* Use look-up-table to get dot density */
    dotdensity = dotdensityLUT[pixelb];

    mod_input = dotdensity + *fm_err_ptr;

    /* Threshold modifed dotdensity */
    thresholding = mod_input - (dbs_pat_rowptr[(j-1)%SCREENWIDTH] * DT + T2);
    output = (thresholding > 0) ? 255 : 0;

    error = output - mod_input;

    e1 = (W1 * error)>>8;
    e2 = (W2 * error)>>8;
    e3 = (W3 * error)>>8;
    /*e4 = (W4 * error)>>8;*/
    e4 = error - e1 - e2 -e3;

    *(++fm_err_ptr) -= e4;
    *(--fm_err_ptr) = fm_tmp - e3;
    *(--fm_err_ptr) -= e1;
    fm_tmp = -e2;

/* Begin section on dot size rendering with partial doting */
    if(output) {
```

```
        /* Left pixel */
        *(img_out_ptr++) = (dotsizeLUT[pixelb]>>1) + NEWRIGHT;
        /* Right pixel */
        if(dotsizeLUT[pixelb] & 1)     /* Take care of quantization error */
           *img_out_ptr = ((dotsizeLUT[pixela]+1)>>1) + NEWLEFT;
        else
           *img_out_ptr = (dotsizeLUT[pixela]>>1) + NEWLEFT;
      }
      else {
        *(img_out_ptr++) = NEWRIGHT;
        *img_out_ptr = NEWLEFT;
      }
      img_out_ptr -= 3;
    }
    *(++img_out_ptr) = NEWLEFT; /* Take care of the first column */
    return;
  }
```

Appendix A2

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* coef.h file */

#define SCREENHEIGHT 128
#define SCREENWIDTH  128

#define NEWRIGHT  0xc0
#define NEWLEFT   0x40
#define NEWCENTER 0x00

#define F1 0x0007    /* Floyd-Steinberg Weights 7/16 in Q4 */
#define F2 0x0003    /* Floyd-Steinberg Weights 3/16 in Q4 */
#define F3 0x0005    /* Floyd-Steinberg Weights 5/16 in Q4 */
#define F4 0x0001    /* Floyd-Steinberg Weights 7/16 in Q4 */

typedef struct TDEDPARA
{
      short T2;
      short DT;
      short W1;
      short W2;
      short W3;
      short W4;
} TDEDPARA;


static TDEDPARA TDEDcoeff[129]={
{76, 0, 181, 0, 3, 72},
{76, 0, 181, 0, 3, 72},
{79, 0, 172, 1, 2, 81},
{80, 0, 161, 14, 18, 63},
{82, 0, 159, 1, 37, 59},
{83, 0, 149, 6, 5, 96},
{83, 0, 141, 30, 0, 85},
{85, 0, 138, 13, 0, 105},
{86, 0, 144, 10, 1, 101},
{85, 0, 129, 48, 3, 76},
{86, 0, 123, 31, 1, 101},
{87, 0, 123, 29, 3, 101},
```

```
{87, 0, 115, 28, 5, 108},
{89, 0, 138, 19, 18, 81},
{89, 0, 111, 17, 51, 77},
{88, 0, 115, 31, 0, 110},
{87, 0, 120, 16, 16, 104},
{88, 0, 139, 12, 0, 105},
{89, 0, 122, 19, 17, 98},
{90, 0, 112, 32, 0, 112},
{91, 0, 98, 34, 20, 104},
{90, 10, 123, 16, 26, 91},
{93, 8, 126, 1, 74, 55},
{89, 10, 89, 26, 71, 70},
{89, 10, 89, 22, 43, 102},
{89, 12, 91, 21, 34, 110},
{88, 12, 85, 24, 30, 117},
{88, 14, 85, 23, 30, 118},
{84, 24, 113, 27, 13, 103},
{82, 26, 113, 33, 0, 110},
{83, 26, 109, 29, 9, 109},
{84, 28, 106, 21, 29, 100},
{85, 28, 103, 13, 56, 84},
{96, 2, 102, 16, 57, 81},
{93, 6, 102, 25, 28, 101},
{91, 12, 102, 24, 32, 98},
{96, 2, 103, 24, 23, 106},
{94, 10, 99, 17, 62, 78},
{95, 6, 110, 12, 110, 24},
{97, 4, 114, 12, 112, 18},
{97, 6, 114, 11, 113, 18},
{96, 8, 111, 14, 110, 21},
{94, 12, 102, 17, 109, 28},
{94, 8, 79, 32, 108, 37},
{95, 6, 74, 35, 110, 37},
{97, 2, 70, 35, 111, 40},
{97, 4, 68, 33, 112, 43},
{97, 6, 69, 28, 112, 47},
{98, 6, 70, 22, 114, 50},
{97, 6, 68, 43, 113, 32},
{100, 4, 68, 22, 114, 52},
{99, 6, 71, 24, 112, 49},
{102, 2, 70, 23, 114, 49},
{100, 6, 68, 23, 114, 51},
{100, 8, 66, 22, 116, 52},
{100, 8, 66, 24, 116, 50},
{96, 16, 75, 0, 122, 59},
{95, 16, 63, 0, 127, 66},
{95, 16, 56, 0, 130, 70},
{97, 14, 56, 0, 132, 68},
{97, 16, 59, 0, 132, 65},
{97, 16, 60, 0, 133, 63},
{98, 16, 62, 0, 133, 61},
{95, 26, 98, 0, 109, 49},
{97, 20, 65, 0, 132, 59},
{98, 18, 61, 0, 132, 63},
{99, 18, 63, 0, 131, 62},
{100, 16, 58, 0, 133, 65},
{100, 16, 58, 0, 131, 67},
```

```
{101, 16, 60, 0, 131, 65},
{101, 16, 63, 0, 129, 64},
{101, 16, 58, 0, 129, 69},
{102, 16, 71, 0, 123, 62},
{103, 8, 68, 23, 114, 51},
{103, 8, 66, 22, 116, 52},
{105, 6, 68, 22, 115, 51},
{106, 4, 70, 22, 114, 50},
{108, 2, 69, 23, 113, 51},
{105, 8, 68, 22, 114, 52},
{108, 6, 70, 20, 115, 51},
{106, 8, 69, 27, 112, 48},
{109, 2, 65, 35, 112, 44},
{110, 4, 69, 34, 111, 42},
{110, 6, 72, 35, 110, 39},
{114, 0, 73, 34, 111, 38},
{110, 12, 94, 21, 108, 33},
{111, 12, 102, 15, 110, 29},
{116, 6, 114, 10, 113, 19},
{96, 16, 92, 16, 67, 81},
{100, 12, 95, 17, 67, 77},
{101, 12, 97, 19, 67, 73},
{99, 4, 101, 20, 45, 90},
{93, 4, 103, 25, 25, 103},
{94, 8, 101, 25, 33, 97},
{78, 24, 99, 26, 19, 112},
{81, 26, 104, 22, 24, 106},
{82, 26, 102, 26, 25, 103},
{91, 26, 109, 14, 46, 87},
{104, 10, 82, 0, 95, 79},
{107, 8, 83, 0, 97, 76},
{105, 8, 87, 2, 84, 83},
{81, 14, 86, 27, 25, 118},
{99, 12, 122, 0, 37, 97},
{102, 10, 117, 0, 45, 94},
{103, 10, 90, 21, 64, 81},
{105, 12, 122, 4, 51, 79},
{101, 12, 126, 9, 29, 92},
{88, 12, 121, 25, 0, 110},
{85, 12, 114, 25, 1, 116},
{89, 10, 109, 23, 10, 114},
{86, 12, 112, 29, 1, 114},
{89, 12, 119, 31, 0, 106},
{94, 10, 123, 37, 1, 95},
{93, 8, 117, 63, 1, 75},
{99, 6, 118, 75, 9, 54},
{97, 6, 120, 43, 3, 90},
{111, 6, 121, 35, 32, 68},
{95, 6, 116, 54, 0, 86},
{107, 6, 125, 39, 15, 77},
{93, 34, 137, 27, 19, 73},
{85, 44, 139, 33, 16, 68},
{87, 48, 146, 31, 23, 56},
{87, 44, 148, 22, 10, 76},
{93, 40, 152, 22, 11, 71},
{97, 44, 159, 4, 28, 65},
{95, 42, 161, 25, 4, 66},
```

```
{103, 48, 176, 3, 44, 33},
{101, 56, 165, 27, 55, 9},
{97, 56, 165, 27, 55, 9},
};

static short OptSizeLUT[256]={
120,
118,
117,
115,
114,
112,
111,
109,
108,
106,
105,
104,
102,
101,
100,
99,
97,
96,
95,
94,
93,
92,
91,
90,
90,
89,
88,
87,
86,
86,
85,
84,
84,
83,
82,
82,
81,
81,
80,
80,
79,
79,
78,
78,
77,
77,
77,
76,
76,
76,
75,
```

75,
74,
74,
74,
74,
73,
73,
73,
72,
72,
72,
72,
71,
71,
71,
71,
70,
70,
70,
70,
70,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
68,
68,
68,
68,
68,
67,
67,
67,
66,
66,
66,
65,
65,
65,
64,
64,
63,
63,
62,
62,
61,
61,
60,
60,
59,
59,
58,
58,
57,
57,
56,
56,
55,
55,
54,
54,
53,
53,
52,
52,
52,
51,
51,
50,
50,

```
49,
49,
48,
48,
48,
47,
47,
46,
46,
46,
45,
45,
44,
44,
44,
43,
43,
43,
42,
42,
42,
42,
41,
41,
41,
40,
40,
40,
39,
39,
39,
38,
38,
38,
};

static short OptDensityLUT[256]={
128,
127,
126,
125,
124,
123,
122,
121,
120,
119,
119,
118,
117,
117,
116,
115,
115,
114,
114,
113,
```

113,
112,
112,
111,
111,
110,
110,
110,
109,
109,
109,
108,
108,
108,
107,
107,
107,
107,
107,
106,
106,
106,
106,
106,
105,
105,
105,
105,
105,
105,
105,
104,
104,
104,
104,
104,
104,
104,
104,
104,
103,
103,
103,
103,
103,
103,
103,
103,
102,
102,
102,
102,
102,
102,
102,
102,
101,

101,
101,
101,
101,
100,
100,
100,
100,
99,
99,
99,
99,
98,
98,
98,
97,
97,
97,
96,
96,
96,
95,
95,
94,
94,
93,
93,
93,
92,
92,
91,
91,
90,
90,
90,
89,
89,
88,
88,
87,
87,
86,
86,
85,
85,
84,
84,
84,
83,
83,
82,
82,
81,
81,
80,
80,
79,

79,
78,
78,
77,
77,
76,
76,
75,
75,
74,
74,
73,
73,
72,
72,
71,
71,
70,
69,
69,
68,
68,
67,
67,
66,
66,
66,
65,
65,
64,
64,
63,
63,
63,
62,
62,
61,
61,
61,
60,
60,
60,
59,
59,
59,
58,
58,
58,
57,
57,
57,
56,
56,
56,
56,
55,
55,

55,
54,
54,
54,
54,
53,
53,
53,
52,
52,
52,
51,
51,
51,
50,
50,
50,
49,
49,
49,
48,
48,
47,
47,
46,
46,
45,
45,
44,
44,
43,
43,
42,
41,
41,
40,
39,
38,
38,
37,
36,
35,
34,
33,
32,
31,
30,
29,
28,
27,
26,
25,
23,
22,
21,
20,
18,

```
17,
15,
14,
12,
10,
8,
6,
0,
};
```

Appendix B1

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* amfm_2bit.c file */

/* 2 bits/pixel am/fm halftoning algorithm: part A */
/* Process input image in 4 strips */
/* Assume width of each stripe is multiple of 8 */
/* One row serpentine TDED */
/* One path amfm with dot size error diffusion */
/* A tiff image file containing bit-packed tokens is generated for amfm_pwm.c */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include "coef.h"
#include "tiff.h"
#include "allocate.h"

void get_stripe(int,int,unsigned char **,int,int,unsigned char **);
void cut_out_result(int,int,int,int,unsigned char **,unsigned char **);
void amfm(unsigned int,unsigned int,unsigned char *,unsigned char *,short *,\
        short *,unsigned char **,TDEDPARA *,TOKENLUT  *,short *,short *);
void amfm_ed_2bits(unsigned int,unsigned int,unsigned char **,unsigned char **,\
                unsigned char **, TDEDPARA *,TOKENLUT  *,short *,short *);
int main(int argc, char ** argv)
{
  int i,j,width,start_point,cut_offset,cut_width,store_offset,stripe_width;
  FILE * fp;
  struct TIFF_img input_img, output_img, mid;
  time_t first, second;
  unsigned char **stripe, **output_stripe;
  TDEDPARA *tdedpara = &TDEDcoeff[0];
  TOKENLUT  *tokenLUT = &TokenLUT[0] + 30;
  short *dotdensityLUT = &OptDensityLUT[0];
  short *dotsizeLUT = &OptSizeLUT[0];


  if(argc<3) {
```

```
        printf("usage: %s input_img.tif output_img.tif\n",argv[0]);
        return 1;
    }

    /* read the input image  */
    if ((fp=fopen(argv[1], "rb"))==NULL) {
      printf("can not open file %s 1\n",argv[1]);
      exit(2);
    }
    if(read_TIFF(fp,&input_img)) {
      printf("error reading input file\n");
      exit(3);
    }
    fclose(fp);

    if((fp=fopen("dbshalf.tif", "rb"))==NULL) {
      fprintf(stderr, "can not open file: dbshalf.tif\n");
      exit(1);
    }
    if(read_TIFF(fp,&mid))
    {
      fprintf(stderr, "error reading file\n");
      exit(1);
    }
    fclose(fp);

    /* Set variable to do timing of algorithm */
    first = time(NULL);

    /* Modify image width to make sure each strip is multiple of 8 */
    width = floor(input_img.width/32.0)*32;

    /* Allocate memory for entire fm output image. */
    get_TIFF( &output_img, input_img.height, width/4, 'g' );

   /* Process 4 stripes independently */

    cut_width = width/16;

    /* first stripe */
    printf("\nProcess first stripe\n");
    stripe_width = width/4+OVERLAP_WIDTH/2;
    stripe = ( unsigned char **
  )multialloc(sizeof(char),2,input_img.height,stripe_width);
    output_stripe = ( unsigned char **
  )multialloc(sizeof(char),2,input_img.height,stripe_width/4);
    start_point = 0;

  get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
  dth,stripe);
    amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
                  mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
    cut_offset = 0;
    store_offset = 0;
    cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
              output_stripe,output_img.mono);
    multifree(stripe,2);
```

```
  multifree(output_stripe,2);
  printf("\n");

  /* Second stripe */
  printf("Process second stripe\n");
  stripe_width = width/4+OVERLAP_WIDTH;
  stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
  output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
  start_point = width/4-OVERLAP_WIDTH/2;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
  amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
                mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  cut_offset = OVERLAP_WIDTH/8;
  store_offset = width/16;
  cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
            output_stripe,output_img.mono);
  multifree(stripe,2);
  multifree(output_stripe,2);
  printf("\n");

  /* Third stripe */
  printf("Process third stripe\n");
  stripe_width = width/4+OVERLAP_WIDTH;
  stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
  output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
  start_point = width/2-OVERLAP_WIDTH/2;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
  amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
                mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  cut_offset = OVERLAP_WIDTH/8;
  store_offset = width/8;
  cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
            output_stripe,output_img.mono);
  multifree(stripe,2);
  multifree(output_stripe,2);
  printf("\n");

  /* Fourth stripe */
  printf("Process fourth stripe\n");
  stripe_width = width/4+OVERLAP_WIDTH/2;
  stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width);
  output_stripe = ( unsigned char **
)multialloc(sizeof(char),2,input_img.height,stripe_width/4);
  start_point = width/4*3-OVERLAP_WIDTH/2;

get_stripe(input_img.height,input_img.width,input_img.mono,start_point,stripe_wi
dth,stripe);
  amfm_ed_2bits(input_img.height,stripe_width,stripe,output_stripe,\
```

```
                mid.mono,tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  cut_offset = OVERLAP_WIDTH/8;
  store_offset = width/16*3;
  cut_out_result(input_img.height,cut_offset,cut_width,store_offset,\
             output_stripe,output_img.mono);
  multifree(stripe,2);
  multifree(output_stripe,2);
  printf("\n");


  /* show the run time */
  second = time(NULL);
  fprintf(stdout,"\nFinished AM/FM and writing results.\n");
  fprintf(stdout,"Cum. run time: %f sec.\n",difftime(second,first));

  /* write PWM codes image */
  if( (fp = fopen(argv[2],"wb"))==NULL) {
    printf ("cannot open file %s\n", argv[2]);
    exit(4);
  }

  if(write_TIFF(fp,&output_img)) {
    printf ("\nError writing TIFF file %s\n", argv[2]);
    return 1;
  }
  fclose(fp);

  /* free the space */
  free_TIFF(&(output_img));
  free_TIFF(&(input_img));
  free_TIFF(&(mid));
  fflush(stdout);
  return 0;
}

void amfm_ed_2bits(
  unsigned int height,              /* Input image height */
  unsigned int width,               /* Input image width */
  unsigned char ** contone_img,     /* Input image [height][width] */
  unsigned char ** token_img, /* Output token image [height][width/4] */
  unsigned char ** dbs_screen,      /* DBS screen used in thresholding of fm
part */
  TDEDPARA *tdedpara,          /* Tone-dependent error diffusion parameters */
  TOKENLUT  *tokenLUT,         /* Token LUT used in dot size diffusion */
  short *dotdensityLUT,        /* Optimal dot density curve */
  short *dotsizeLUT)           /* Optimal dot size curve */
{
  short *fm_err,*am_err;
  unsigned int i,j,token_img_width, mod_height;

  /* Initialize first row of fm error buffer */
  srand(1); /* fix the seed */
  fm_err = (short*)malloc(sizeof(short) * (width+2));
  for(j = 0; j<width+2; j++)
    fm_err[j] = (rand()%128-64);  /* Initialization */

  /* initialize first row of am (dot size) error buffer */
```

```
  am_err = (short*)malloc(sizeof(short) * ((width>>1)+4));
  for(i = 0; i<(width>>1)+4; i++)
     am_err[i]=0;        /* Initialization */

  /* Avoid boundary because of pairwise row process */
  if(height & 1)
     mod_height = height - 1;
  else
     mod_height = height;

  /* Process the input image with 2 rows each time */
  for(i=0; i<mod_height; i+=2) {
    if((i%600) == 0) printf("amfm_ed: starting row %d\n", i);
    amfm(width,i,contone_img[i],token_img[i],fm_err,am_err,dbs_screen,\
         tdedpara,tokenLUT,dotdensityLUT,dotsizeLUT);
  }

  /* Take care the last row if necessary */
  if(height & 1)
  {
    token_img_width = width/4;
    for(j=0;j<token_img_width;j++)
       *(token_img[mod_height]+j) = 0;
  }

  free(fm_err);
  free(am_err);
  return;

}

/* Define macro of FM_EVEN_ROW which does fm for a pair pixels in even row */
#define FM_EVEN_ROW                                                    \
    /* First process FM (dot density) for left pixel in pixel pair. */        \
                                                                       \
    /* Get first pixel */                                              \
    pixela = *(img_in_ptr++);                                          \
                                                                       \
    /* Use look-up-table to get dot density */                             \
    dotdensity = dotdensityLUT[pixela];                                \
                                                                       \
    /* Compute look-up table entries for tone dependent error diffusion */    \
    tded_ptr = (short*)(tdedpara + dotdensity);                            \
    T2 = tded_ptr[0];                                                  \
    DT = tded_ptr[1];                                                  \
    W1 = tded_ptr[2];                                                  \
    W2 = tded_ptr[3];                                                  \
    W3 = tded_ptr[4];                                                  \
    W4 = tded_ptr[5];                                                  \
                                                                       \
    /* compute dotdensity modified by diffused error */                    \
    mod_input = dotdensity + *fm_err_ptr;                              \
                                                                       \
    /* Threshold modifed dotdensity */                                 \
    thresholding = mod_input - (dbs_pat_rowptr[j%SCREENWIDTH] * DT + T2);      \
    output = (thresholding > 0) ? 255 : 0;                             \
                                                                       \
```

```
    /* Compute weighted errors */                                          \
    error = output - mod_input;                                                \
    e1 = (W1 * error)>>8;                                                  \
    e2 = (W2 * error)>>8;                                                  \
    e3 = (W3 * error)>>8;                                                  \
    /*e4 = (W4 * error)>>8;*/                                              \
    e4 = error - e1 - e2 -e3;                                              \
    /* Diffuse error forward in 1-D error buffer */                            \
    *(--fm_err_ptr) -= e4;                                                 \
    *(++fm_err_ptr) = fm_tmp - e3;                                         \
    *(++fm_err_ptr) -= e1;                                                 \
    fm_tmp = -e2;                                                      \
                                                                       \
    /* Now process FM (dot density) for right pixel in pixel pair. */         \
    /* Use same TDED parameters as for left pixel. */                          \
                                                                       \
    /* Get second pixel */                                                 \
    pixelb = *(img_in_ptr++);                                              \
                                                                       \
    /* Use look-up-table to get dot density */                             \
    dotdensity = dotdensityLUT[pixelb];                                    \
                                                                       \
    mod_input = dotdensity + *fm_err_ptr;                                  \
    error = - mod_input;  /* suppress dot firing at this pixel */          \
                                                                       \
    e1 = (W1 * error)>>8;                                                  \
    e2 = (W2 * error)>>8;                                                  \
    e3 = (W3 * error)>>8;                                                  \
    /*e4 = (W4 * error)>>8;*/                                              \
    e4 = error - e1 - e2 -e3;                                              \
    /* Using the tded weights of the left pixel */                             \
    *(--fm_err_ptr) -= e4;                                                 \
    *(++fm_err_ptr) = fm_tmp - e3;                                         \
    *(++fm_err_ptr) -= e1;                                                 \
    fm_tmp = -e2;                                                      \
    j += 2;

/* Define macro of AM_ERROR_EVEN_ROW which computes and distributes
   dot size error for a pair pixels in even row */
#define AM_ERROR_EVEN_ROW                                                  \
    /*      e1 = F1 * error;  */                                             \
    e2 = F2 * error;                                                       \
    e3 = F3 * error;                                                       \
    e4 = (F4 * error);                                                     \
    e1 = error*16 - e2 -e3 -e4;                                            \
    am_err_ptr -= 2;                                                       \
    *(am_err_ptr) -=e4;                                                    \
    *(++am_err_ptr) -=e3;                                                  \
    *(++am_err_ptr) = -e2;                                                 \
    *(++am_err_ptr) -=e1;

/* Define macro of FM_ODD_ROW which does fm for a pair pixels in odd row */
#define FM_ODD_ROW                                                         \
  /* First process FM (dot density) for right pixel in pixel pair */           \
                                                                       \
  /* Get right pixel */                                                    \
    pixela = *(img_in_ptr--);                                              \
```

```
                                                                    \
/* Use look-up-table to get dot density */                          \
dotdensity = dotdensityLUT[pixela];                                 \
                                                                    \
/* Compute look-up table entries for tone dependent error diffusion */    \
tded_ptr = (short*)(tdedpara + dotdensity);                         \
                                                                    \
T2 = tded_ptr[0];                                                       \
DT = tded_ptr[1];                                                       \
W1 = tded_ptr[2];                                                       \
W2 = tded_ptr[3];                                                       \
W3 = tded_ptr[4];                                                       \
W4 = tded_ptr[5];                                                       \
                                                                    \
/* Compute dotdensity modified by diffused error */                         \
mod_input = dotdensity + *fm_err_ptr;                                   \
                                                                    \
/* suppress this dot and compute the error */                           \
error = - mod_input;                                                    \
                                                                    \
/* Compute weighted errors */                                           \
e1 = (W1 * error)>>8;                                                   \
e2 = (W2 * error)>>8;                                                   \
e3 = (W3 * error)>>8;                                                   \
/*e4 = ((W4 * error)>>8);*/                                             \
e4 = error - e1 - e2 -e3;                                               \
                                                                        \
/* Diffuse error forward in 1-D error buffer */                         \
*(++fm_err_ptr) -= e4;                                                  \
*(--fm_err_ptr) = fm_tmp - e3;                                          \
*(--fm_err_ptr) -= e1;                                                  \
fm_tmp = -e2;                                                   \
                                                                \
/* Now process FM (dot density) for Left pixel in a pair */             \
                                                                        \
/* Get second pixel */                                                  \
pixelb = *(img_in_ptr--);                                               \
                                                                \
/* Use look-up-table to get dot density */                              \
dotdensity = dotdensityLUT[pixelb];                                     \
                                                                        \
mod_input = dotdensity + *fm_err_ptr;                                   \
                                                                \
/* Threshold modifed dot density */                                     \
thresholding = mod_input - (dbs_pat_rowptr[(j-1)%SCREENWIDTH] * DT + T2); \
output = (thresholding > 0) ? 255 : 0;                                  \
                                                                \
j -= 2;                                                         \
                                                                \
error = output - mod_input;                                             \
                                                                        \
e1 = (W1 * error)>>8;                                                   \
e2 = (W2 * error)>>8;                                                   \
e3 = (W3 * error)>>8;                                                   \
/*e4 = (W4 * error)>>8;*/                                               \
e4 = error - e1 - e2 -e3;                                               \
                                                                        \
```

```
    *(++fm_err_ptr) -= e4;                                          \
    *(--fm_err_ptr) = fm_tmp - e3;                                  \
    *(--fm_err_ptr) -= e1;                                          \
    fm_tmp = -e2;

/* Define macro of AM_ERROR_ODD_ROW which computes and distributes
   dot size error for a pair pixels in odd row */
#define AM_ERROR_ODD_ROW                                            \
    /*      e1 = F1 * error;  */                                    \
    e2 = F2 * error;                                                \
    e3 = F3 * error;                                                \
    e4 = F4 * error;                                                \
    e1 = error*16 - e2 -e3 -e4;                                     \
                                                                    \
    am_err_ptr += 2;                                                \
    *am_err_ptr -= e4;                                              \
    *(--am_err_ptr) -= e3;                                          \
    *(--am_err_ptr) = -e2;                                          \
    *(--am_err_ptr) -= e1;

/* This subroutine only processes 2 rows */
/* Assume width of image is multiple of 8 */
void amfm(
  unsigned int width,    /* Input image width */
  unsigned int i,        /* ith row */
  unsigned char *img_in,      /* ith row of input image array */
  unsigned char *img_out,     /* ith row of output image array */
  short *fm_err,         /* FM error buffer */
  short *am_err,         /* AM error buffer */
  unsigned char ** dbs_screen,      /* dbs_screen[SCREENHEIGHT][SCREENWIDTH] */
  TDEDPARA *tdedpara,    /* Tone-dependent error diffusion parameters */
  TOKENLUT  *tokenLUT,   /* Token look-up-table used in dot size diffusion */
  short *dotdensityLUT, /* Optimal dot density curve */
  short *dotsizeLUT)     /* Optimal dot size curve  */

{
  short fm_tmp,thresholding;
  short *fm_err_ptr, *am_err_ptr;
  short pixela, pixelb, mod_dotsize, output;
  unsigned int j, img_out_width;
  unsigned char bit_pack;
  unsigned char *img_in_ptr, *img_out_ptr, *dbs_pat_rowptr;
  short dotdensity, dotsize, mod_input, error;
  short W1, W2, W3, W4, T2, DT, e1, e2, e3, e4;
  short *tded_ptr, *token_lut_ptr;
  FILE *fp;


  /*-------------------------------------*/
  /* serpentine even rows                */
  /*-------------------------------------*/
      /* Initial points */
  fm_tmp = 0;
  fm_err_ptr = fm_err+1;
  am_err_ptr = am_err+2;
  img_in_ptr = img_in;
  img_out_ptr = img_out;
```

```
/* Get row pointer of dbs pattern */
/* SCREENHEIGHT = 128, module '(i++)%128' can be replace by '(i++)&127' */
dbs_pat_rowptr = dbs_screen[(i++)%SCREENHEIGHT];

/* Index through pixels in pairs */
for(j = 0; j<width;) {

  /* FM halftoning for first pixel pair */
  FM_EVEN_ROW

  /* Begin section on AM halftoning for first pixel pair. */
  /* This section computes 2-bit PWM codes for dot pairs. */
  /* Errors in AM component are diffused using Floyd-Steinberg weights. */
  /* Tokens of left and right pixels along with size error are precomputed */
  /* and stored in tokenLUT */

  /* Get diffused error from dot size error buffer */
  /* This operation can be replace by y=x/16 without affecting */
  /* the quality of the halftone. */
  mod_dotsize = (*am_err_ptr+8)>>4;

  bit_pack = 0;
  if(output) {
    mod_dotsize += dotsizeLUT[pixela];
    /* Get 2 PWM tokens and error corresponding to mod_dotsize */
    /* Then pack the tokens */
    token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
    bit_pack = token_lut_ptr[0]<<6;     /* Get and pack token for left pixel
*/
    bit_pack += token_lut_ptr[1]<<4;    /* Get and pack token for right pixel
*/
    error = token_lut_ptr[2];           /* Get size error for the pixel pair
*/
  }
  else
    error = - mod_dotsize;

  /* Compute and distribute dot size error */
  AM_ERROR_EVEN_ROW

  /* FM halftoning for second pixel pair */
  FM_EVEN_ROW

  /* Begin section on AM halftoning for second pixel pair. */
  /* Same comments for AM halftoning of first pixel pair */
  mod_dotsize = (*am_err_ptr+8)>>4;
  if(output) {
    mod_dotsize += dotsizeLUT[pixela];

    /* Get 2 PWM tokens and error corresponding to mod_dotsize */
    /* Then pack the tokens */
    token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
    bit_pack += token_lut_ptr[0]<<2;    /* Get and pack token for left pixel
*/
    bit_pack += token_lut_ptr[1]; /* Get and pack token for right pixel */
```

```
      error = token_lut_ptr[2];           /* Get size error for the pixel pair
*/
    }
    else
      error = - mod_dotsize;

    /* Compute and distribute dot size error */
    AM_ERROR_EVEN_ROW
    /* write the packed tokens to output image array */
    *(img_out_ptr++) = bit_pack;

  }   /* end of ith row */


/*-----------------------------------*/
/* serpentine odd rows               */
/*-----------------------------------*/
  img_out_width = width/4;
  fm_tmp = 0;
  /* Set fm error buffer pointer to the end of fm_err buffer */
  fm_err_ptr = fm_err+ width - 1;  /* Offset by 1 */
  /* Set am error buffer pointer to the end of am_err buffer */
  am_err_ptr = am_err + (width>>1);  /* Offset by 2 */
  img_in_ptr = img_in+width*2-2;
  img_out_ptr = img_out+img_out_width*2-1;

  /* Get row pointer of dbs pattern */
  /* SCREENHEIGHT = 128, module 'i%128' can be replace by 'i&127' */
  dbs_pat_rowptr = dbs_screen[i%SCREENHEIGHT];

  /* Index through pixels in pairs */
  bit_pack = 0;
  for(j = width-2; j>2;) {

    /* FM halftoning for first pixel pair */
    FM_ODD_ROW

    /* Begin section on AM halftoning for the first pixel pairl */
    /* This section computes 2-bit PWM codes for dot pairs. */
    /* Errors in AM component are diffused using Floyd-Steinberg weights. */
    /* Tokens of left and right pixels along with size error are precomputed */
    /* and stored in tokenLUT */

    /* Get diffused error from dot size error buffer */
    /* This operation can be replace by y=x/16 without affecting */
    /* the quality of the halftone. */
    mod_dotsize = (*am_err_ptr+8)>>4;
    if(output)
    {
      mod_dotsize += dotsizeLUT[pixelb];
      /* Get 2 PWM tokens and error corresponding to mod_dotsize */
      /* Then pack the tokens */
      token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
      error = token_lut_ptr[2];     /* Get size error for the pixel pair */
      bit_pack += token_lut_ptr[1]<<2;    /* Get and pack token for right pixel
 */
```

```
      bit_pack += token_lut_ptr[0]<<4;    /* Get and pack token for left pixel
*/
    }
    else
      error = - mod_dotsize;
    /* Compute and distribute dot size error */
    AM_ERROR_ODD_ROW

    /* FM halftoning for second pixel pair */
    FM_ODD_ROW

    /* Section on AM halftoning for second pixel pair */
    /* Same comments as on AM halftong for the first pixel pair */
    mod_dotsize = (*am_err_ptr+8)>>4;
    if(output)
    {
      mod_dotsize += dotsizeLUT[pixelb];
      /* Get 2 PWM tokens and error corresponding to mod_dotsize */
      /* Then pack it */
      token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
      error = token_lut_ptr[2];     /* Get size error for the pixel pair */
      bit_pack += token_lut_ptr[1]<<6; /* Get and pack token for right pixel */

      /* Write the packed tokens to the output image array */
      *(img_out_ptr--) = bit_pack;

      bit_pack = token_lut_ptr[0];  /* Get and pack token for left pixel */
    }
    else
    {
      *(img_out_ptr--) = bit_pack;
      bit_pack = 0;
      error = - mod_dotsize;
    }
    /* Compute and distribute dot size error */
    AM_ERROR_ODD_ROW
  }
  /* Take care of the most left three pixels of odd rows */

  /* FM halftoning for the first pixel pair */
  FM_ODD_ROW

  /* AM halftoning for first pixel pair */
  mod_dotsize = (*am_err_ptr+8)>>4;
  if(output)
  {
    mod_dotsize += dotsizeLUT[pixelb];

    /* Get 2 PWM tokens and error corresponding to mod_dotsize */
    /* Then pack the tokens */
    token_lut_ptr = (short *) (tokenLUT + mod_dotsize);
    error = token_lut_ptr[2]; /* Get size error for the pixel pair */
    bit_pack += token_lut_ptr[1]<<2;      /* Get token for right pixel */
    bit_pack += token_lut_ptr[0]<<4;      /* Get token for left pixel */
  }
  else
    error = - mod_dotsize;
```

```
  /* Write the packed tokens to the output image array */
  *(img_out_ptr--) = bit_pack;

  /* Compute and distribute dot size error */
  AM_ERROR_ODD_ROW

  return;
}

void get_stripe(
  int img_height,
  int img_width,
  unsigned char **contone_img,
  int start_point,
  int stripe_width,
  unsigned char **stripe)
{
  int i,j;

  for(i=0;i<img_height;i++)
  for(j=0;j<stripe_width;j++)
      stripe[i][j] = contone_img[i][j+start_point];
}

void cut_out_result(
  int img_height,
  int cut_offset,
  int cut_width,
  int store_offset,
  unsigned char **output_stripe,
  unsigned char **output_img)
{
  int i,j;
  for(i=0;i<img_height;i++)
  for(j=0;j<cut_width;j++)
    output_img[i][j+store_offset]=output_stripe[i][j+cut_offset];
}
```

Appendix B2

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* amfm_convert.c file */

/* 2 bits/pixel am/fm halftoning algorithm: part B */
/* Input a tiff file containing 2-bit tokens */
/* Output a tiff file containing pulse width modulation codes */
/* Every pixel is left justified */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#define NEWRIGHT  0xc0
#define NEWLEFT   0x40
#define NEWCENTER 0x00
void token2pwm(int,int,unsigned char **,unsigned char **,unsigned char *);
int main(int argc, char ** argv)
{
  unsigned char pwm[4]={0,21,42,63};
  int i,j;
  FILE * fp;
  struct TIFF_img input_img,output_img;

  if(argc<3) {
      printf("usage: %s token.tif output_img.tif\n",argv[0]);
      return 1;
  }

  printf("Mapping tokens to pulse width modulation codes.\n");

  /* read the input image  */
  if ((fp=fopen(argv[1], "rb"))==NULL) {
    printf("can not open file %s 1\n",argv[1]);
    exit(2);
  }
  if(read_TIFF(fp,&input_img)) {
    printf("error reading input file\n");
    exit(3);
```

```
  }
  fclose(fp);

    /* Allocate memory for entire fm output image. */
  get_TIFF( &output_img, input_img.height, input_img.width*4, 'g' );

  token2pwm(input_img.height, input_img.width, input_img.mono, output_img.mono,
pwm);

  /* write PWM codes image */
  if( (fp = fopen(argv[2],"wb"))==NULL) {
    printf ("cannot open file %s\n", argv[3]);
    exit(5);
  }

  if(write_TIFF(fp,&output_img)) {
    printf ("\nError writing TIFF file %s\n", argv[2]);
    return 1;
  }
  fclose(fp);

  /* free the space */
  free_TIFF(&(input_img));
  free_TIFF(&(output_img));
  fflush(stdout);
  return 0;
}



/* Map tokens to pulse width modulation codes */
void token2pwm(
int img_height,
int img_width,
unsigned char ** token,
unsigned char ** output_pwm,
unsigned char * pwm
)
{
  int i,j,k;

  for(i=0;i<img_height;i++)
      for(j=0;j<img_width;j++)
      {
      output_pwm[i][j*4]   = pwm[(token[i][j]&192)/64] + NEWLEFT;
        output_pwm[i][j*4+1] = pwm[(token[i][j]&48)/16] + NEWLEFT;
        output_pwm[i][j*4+2] = pwm[(token[i][j]&12)/4] + NEWLEFT;
        output_pwm[i][j*4+3] = pwm[token[i][j]&3] + NEWLEFT;
      }
}
```

Appendix B3

COMBINED DOT DENSITY AND DOT SIZE MODULATION

Zhen He
Charles A Bouman
Qian Lin

10003284
M-8658 US


```
/* coef.h file */
#define SCREENHEIGHT 128      /* DBS screen height */
#define SCREENWIDTH  128      /* DBS screen width */
#define OVERLAP_WIDTH 16      /* Width of overlapping region */
#define F1 0x0007 /* Floyd-Steinberg Weights 1/16 in Q4 */
#define F2 0x0003 /* Floyd-Steinberg Weights 5/16 in Q4 */
#define F3 0x0005 /* Floyd-Steinberg Weights 3/16 in Q4 */
#define F4 0x0001 /* Floyd-Steinberg Weights 7/16 in Q4 */
typedef struct TDEDPARA
{
short T2;
short DT;
short W1;
short W2;
short W3;
short W4;
} TDEDPARA;
typedef struct TOKENLUT
{
short left;
short right;
short size_error;
} TOKENLUT;
static TDEDPARA TDEDcoeff[256]={
{76, 0, 181, 0, 3, 72},
{76, 0, 181, 0, 3, 72},
{79, 0, 172, 1, 2, 81},
{80, 0, 161, 14, 18, 63},
{82, 0, 159, 1, 37, 59},
{83, 0, 149, 6, 5, 96},
{83, 0, 141, 30, 0, 85},
{85, 0, 138, 13, 0, 105},
{86, 0, 144, 10, 1, 101},
{85, 0, 129, 48, 3, 76},
{86, 0, 123, 31, 1, 101},
{87, 0, 123, 29, 3, 101},
{87, 0, 115, 28, 5, 108},
```

```
{89, 0, 138, 19, 18, 81},
{89, 0, 111, 17, 51, 77},
{88, 0, 115, 31, 0, 110},
{87, 0, 120, 16, 16, 104},
{88, 0, 139, 12, 0, 105},
{89, 0, 122, 19, 17, 98},
{90, 0, 112, 32, 0, 112},
{91, 0, 98, 34, 20, 104},
{90, 10, 123, 16, 26, 91},
{93, 8, 126, 1, 74, 55},
{89, 10, 89, 26, 71, 70},
{89, 10, 89, 22, 43, 102},
{89, 12, 91, 21, 34, 110},
{88, 12, 85, 24, 30, 117},
{88, 14, 85, 23, 30, 118},
{84, 24, 113, 27, 13, 103},
{82, 26, 113, 33, 0, 110},
{83, 26, 109, 29, 9, 109},
{84, 28, 106, 21, 29, 100},
{85, 28, 103, 13, 56, 84},
{96, 2, 102, 16, 57, 81},
{93, 6, 102, 25, 28, 101},
{91, 12, 102, 24, 32, 98},
{96, 2, 103, 24, 23, 106},
{94, 10, 99, 17, 62, 78},
{95, 6, 110, 12, 110, 24},
{97, 4, 114, 12, 112, 18},
{97, 6, 114, 11, 113, 18},
{96, 8, 111, 14, 110, 21},
{94, 12, 102, 17, 109, 28},
{94, 8, 79, 32, 108, 37},
{95, 6, 74, 35, 110, 37},
{97, 2, 70, 35, 111, 40},
{97, 4, 68, 33, 112, 43},
{97, 6, 69, 28, 112, 47},
{98, 6, 70, 22, 114, 50},
{97, 6, 68, 43, 113, 32},
{100, 4, 68, 22, 114, 52},
{99, 6, 71, 24, 112, 49},
{102, 2, 70, 23, 114, 49},
{100, 6, 68, 23, 114, 51},
{100, 8, 66, 22, 116, 52},
{100, 8, 66, 24, 116, 50},
{96, 16, 75, 0, 122, 59},
{95, 16, 63, 0, 127, 66},
{95, 16, 56, 0, 130, 70},
{97, 14, 56, 0, 132, 68},
{97, 16, 59, 0, 132, 65},
{97, 16, 60, 0, 133, 63},
{98, 16, 62, 0, 133, 61},
{95, 26, 98, 0, 109, 49},
{97, 20, 65, 0, 132, 59},
{98, 18, 61, 0, 132, 63},
{99, 18, 63, 0, 131, 62},
{100, 16, 58, 0, 133, 65},
{100, 16, 58, 0, 131, 67},
{101, 16, 60, 0, 131, 65},
```

```
{101, 16, 63, 0, 129, 64},
{101, 16, 58, 0, 129, 69},
{102, 16, 71, 0, 123, 62},
{103, 8, 68, 23, 114, 51},
{103, 8, 66, 22, 116, 52},
{105, 6, 68, 22, 115, 51},
{106, 4, 70, 22, 114, 50},
{108, 2, 69, 23, 113, 51},
{105, 8, 68, 22, 114, 52},
{108, 6, 70, 20, 115, 51},
{106, 8, 69, 27, 112, 48},
{109, 2, 65, 35, 112, 44},
{110, 4, 69, 34, 111, 42},
{110, 6, 72, 35, 110, 39},
{114, 0, 73, 34, 111, 38},
{110, 12, 94, 21, 108, 33},
{111, 12, 102, 15, 110, 29},
{116, 6, 114, 10, 113, 19},
{96, 16, 92, 16, 67, 81},
{100, 12, 95, 17, 67, 77},
{101, 12, 97, 19, 67, 73},
{99, 4, 101, 20, 45, 90},
{93, 4, 103, 25, 25, 103},
{94, 8, 101, 25, 33, 97},
{78, 24, 99, 26, 19, 112},
{81, 26, 104, 22, 24, 106},
{82, 26, 102, 26, 25, 103},
{91, 26, 109, 14, 46, 87},
{104, 10, 82, 0, 95, 79},
{107, 8, 83, 0, 97, 76},
{105, 8, 87, 2, 84, 83},
{81, 14, 86, 27, 25, 118},
{99, 12, 122, 0, 37, 97},
{102, 10, 117, 0, 45, 94},
{103, 10, 90, 21, 64, 81},
{105, 12, 122, 4, 51, 79},
{101, 12, 126, 9, 29, 92},
{88, 12, 121, 25, 0, 110},
{85, 12, 114, 25, 1, 116},
{89, 10, 109, 23, 10, 114},
{86, 12, 112, 29, 1, 114},
{89, 12, 119, 31, 0, 106},
{94, 10, 123, 37, 1, 95},
{93, 8, 117, 63, 1, 75},
{99, 6, 118, 75, 9, 54},
{97, 6, 120, 43, 3, 90},
{111, 6, 121, 35, 32, 68},
{95, 6, 116, 54, 0, 86},
{107, 6, 125, 39, 15, 77},
{93, 34, 137, 27, 19, 73},
{85, 44, 139, 33, 16, 68},
{87, 48, 146, 31, 23, 56},
{87, 44, 148, 22, 10, 76},
{93, 40, 152, 22, 11, 71},
{97, 44, 159, 4, 28, 65},
{95, 42, 161, 25, 4, 66},
{103, 48, 176, 3, 44, 33},
```

```
{101, 56, 165, 27, 55, 9},
{97, 56, 165, 27, 55, 9},
{103, 48, 176, 3, 44, 33},
{117, 42, 161, 25, 4, 66},
{113, 44, 159, 4, 28, 65},
{121, 40, 152, 22, 11, 71},
{123, 44, 148, 22, 10, 76},
{119, 48, 146, 31, 23, 56},
{125, 44, 139, 33, 16, 68},
{127, 34, 137, 27, 19, 73},
{141, 6, 125, 39, 15, 77},
{153, 6, 116, 54, 0, 86},
{137, 6, 121, 35, 32, 68},
{151, 6, 120, 43, 3, 90},
{149, 6, 118, 75, 9, 54},
{153, 8, 117, 63, 1, 75},
{150, 10, 123, 37, 1, 95},
{153, 12, 119, 31, 0, 106},
{156, 12, 112, 29, 1, 114},
{155, 10, 109, 23, 10, 114},
{157, 12, 114, 25, 1, 116},
{154, 12, 121, 25, 0, 110},
{141, 12, 126, 9, 29, 92},
{137, 12, 122, 4, 51, 79},
{141, 10, 90, 21, 64, 81},
{142, 10, 117, 0, 45, 94},
{143, 12, 122, 0, 37, 97},
{159, 14, 86, 27, 25, 118},
{141, 8, 87, 2, 84, 83},
{139, 8, 83, 0, 97, 76},
{140, 10, 82, 0, 95, 79},
{137, 26, 109, 14, 46, 87},
{146, 26, 102, 26, 25, 103},
{147, 26, 104, 22, 24, 106},
{152, 24, 99, 26, 19, 112},
{152, 8, 101, 25, 33, 97},
{157, 4, 103, 25, 25, 103},
{151, 4, 101, 20, 45, 90},
{141, 12, 97, 19, 67, 73},
{142, 12, 95, 17, 67, 77},
{142, 16, 92, 16, 67, 81},
{132, 6, 114, 10, 113, 19},
{131, 12, 102, 15, 110, 29},
{132, 12, 94, 21, 108, 33},
{140, 0, 73, 34, 111, 38},
{138, 6, 72, 35, 110, 39},
{140, 4, 69, 34, 111, 42},
{143, 2, 65, 35, 112, 44},
{140, 8, 69, 27, 112, 48},
{140, 6, 70, 20, 115, 51},
{141, 8, 68, 22, 114, 52},
{144, 2, 69, 23, 113, 51},
{144, 4, 70, 22, 114, 50},
{143, 6, 68, 22, 115, 51},
{143, 8, 66, 22, 116, 52},
{143, 8, 68, 23, 114, 51},
{136, 16, 71, 0, 123, 62},
```

```
{137, 16, 58, 0, 129, 69},
{137, 16, 63, 0, 129, 64},
{137, 16, 60, 0, 131, 65},
{138, 16, 58, 0, 131, 67},
{138, 16, 58, 0, 133, 65},
{137, 18, 63, 0, 131, 62},
{138, 18, 61, 0, 132, 63},
{137, 20, 65, 0, 132, 59},
{133, 26, 98, 0, 109, 49},
{140, 16, 62, 0, 133, 61},
{141, 16, 60, 0, 133, 63},
{141, 16, 59, 0, 132, 65},
{143, 14, 56, 0, 132, 68},
{143, 16, 56, 0, 130, 70},
{143, 16, 63, 0, 127, 66},
{142, 16, 75, 0, 122, 59},
{146, 8, 66, 24, 116, 50},
{146, 8, 66, 22, 116, 52},
{148, 6, 68, 23, 114, 51},
{150, 2, 70, 23, 114, 49},
{149, 6, 71, 24, 112, 49},
{150, 4, 68, 22, 114, 52},
{151, 6, 68, 43, 113, 32},
{150, 6, 70, 22, 114, 50},
{151, 6, 69, 28, 112, 47},
{153, 4, 68, 33, 112, 43},
{155, 2, 70, 35, 111, 40},
{153, 6, 74, 35, 110, 37},
{152, 8, 79, 32, 108, 37},
{148, 12, 102, 17, 109, 28},
{150, 8, 111, 14, 110, 21},
{151, 6, 114, 11, 113, 18},
{153, 4, 114, 12, 112, 18},
{153, 6, 110, 12, 110, 24},
{150, 10, 99, 17, 62, 78},
{156, 2, 103, 24, 23, 106},
{151, 12, 102, 24, 32, 98},
{155, 6, 102, 25, 28, 101},
{156, 2, 102, 16, 57, 81},
{141, 28, 103, 13, 56, 84},
{142, 28, 106, 21, 29, 100},
{145, 26, 109, 29, 9, 109},
{146, 26, 113, 33, 0, 110},
{146, 24, 113, 27, 13, 103},
{152, 14, 85, 23, 30, 118},
{154, 12, 85, 24, 30, 117},
{153, 12, 91, 21, 34, 110},
{155, 10, 89, 22, 43, 102},
{155, 10, 89, 26, 71, 70},
{153, 8, 126, 1, 74, 55},
{154, 10, 123, 16, 26, 91},
{163, 0, 98, 34, 20, 104},
{164, 0, 112, 32, 0, 112},
{165, 0, 122, 19, 17, 98},
{166, 0, 139, 12, 0, 105},
{167, 0, 120, 16, 16, 104},
{166, 0, 115, 31, 0, 110},
```

```
{165, 0, 111, 17, 51, 77},
{165, 0, 138, 19, 18, 81},
{167, 0, 115, 28, 5, 108},
{167, 0, 123, 29, 3, 101},
{168, 0, 123, 31, 1, 101},
{169, 0, 129, 48, 3, 76},
{168, 0, 144, 10, 1, 101},
{169, 0, 138, 13, 0, 105},
{171, 0, 141, 30, 0, 85},
{171, 0, 149, 6, 5, 96},
{172, 0, 159, 1, 37, 59},
{174, 0, 161, 14, 18, 63},
{175, 0, 172, 1, 2, 81},
{178, 0, 181, 0, 3, 72},
{178, 0, 181, 0, 3, 72},
};
static short OptSizeLUT[256]={
120,
118,
117,
116,
115,
114,
112,
111,
109,
108,
107,
105,
104,
102,
101,
100,
100,
98,
97,
97,
96,
95,
94,
93,
93,
92,
91,
90,
89,
89,
88,
87,
86,
86,
85,
84,
84,
83,
82,
82,
```

81,
80,
80,
79,
79,
78,
78,
78,
77,
77,
77,
76,
76,
76,
75,
75,
75,
75,
74,
74,
74,
73,
73,
72,
72,
72,
71,
71,
71,
70,
70,
70,
70,
70,
70,
70,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,

69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
69,
68,
68,
68,
68,
67,
67,
67,
67,
66,
66,
65,
65,
65,
64,
64,
63,
63,
62,
62,
61,
61,
60,
60,
59,
59,
58,
58,
57,
57,
56,
56,
55,
55,

```
54,
54,
53,
53,
52,
52,
52,
51,
51,
50,
50,
49,
49,
48,
48,
47,
47,
47,
46,
46,
46,
45,
45,
44,
44,
44,
43,
43,
43,
42,
42,
42,
42,
41,
41,
41,
40,
40,
40,
39,
39,
38,
38,
38,
38,
};
static short OptDensityLUT[256]={
128,
125,
123,
121,
119,
117,
116,
115,
114,
113,
```

113,
112,
112,
111,
111,
111,
111,
111,
110,
110,
110,
110,
110,
110,
109,
109,
109,
109,
109,
109,
109,
108,
108,
108,
108,
108,
108,
108,
108,
107,
107,
107,
107,
107,
107,
107,
107,
107,
107,
107,
107,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
106,
105,
105,
105,

105,
105,
105,
104,
104,
104,
104,
104,
103,
103,
103,
102,
102,
102,
101,
101,
101,
100,
100,
100,
99,
99,
98,
98,
98,
97,
97,
96,
96,
96,
95,
95,
94,
94,
93,
93,
93,
92,
92,
91,
91,
91,
90,
90,
89,
89,
88,
88,
88,
87,
87,
86,
86,
85,
85,
84,
84,

83,
83,
83,
82,
82,
81,
81,
80,
80,
79,
79,
78,
78,
77,
77,
76,
76,
75,
75,
74,
74,
73,
73,
72,
71,
71,
70,
70,
69,
69,
68,
68,
67,
67,
66,
66,
65,
65,
64,
64,
64,
63,
63,
62,
62,
62,
61,
61,
61,
60,
60,
60,
60,
59,
59,
59,
59,

59,
58,
58,
58,
58,
58,
58,
58,
58,
57,
57,
57,
57,
57,
57,
57,
57,
57,
57,
57,
57,
56,
56,
56,
56,
56,
56,
56,
56,
56,
56,
56,
55,
55,
55,
55,
55,
55,
54,
54,
54,
54,
53,
53,
53,
52,
52,
52,
51,
51,
50,
50,
49,
48,
48,
47,
46,
45,
44,

```
43,
42,
41,
40,
39,
37,
36,
35,
33,
31,
29,
27,
24,
20,
16,
11,
7,
0,
};
static TOKENLUT TokenLUT[180]={
{0, 0, 30}, /* -30 */
{0, 0, 29}, /* -29 */
{0, 0, 28}, /* -28 */
{0, 0, 27}, /* -27 */
{0, 0, 26}, /* -26 */
{0, 0, 25}, /* -25 */
{0, 0, 24}, /* -24 */
{0, 0, 23}, /* -23 */
{0, 0, 22}, /* -22 */
{0, 0, 21}, /* -21 */
{0, 0, 20}, /* -20 */
{0, 0, 19}, /* -19 */
{0, 0, 18}, /* -18 */
{0, 0, 17}, /* -17 */
{0, 0, 16}, /* -16 */
{0, 0, 15}, /* -15 */
{0, 0, 14}, /* -14 */
{0, 0, 13}, /* -13 */
{0, 0, 12}, /* -12 */
{0, 0, 11}, /* -11 */
{0, 0, 10}, /* -10 */
{0, 0, 9},  /* -9 */
{0, 0, 8},  /* -8 */
{0, 0, 7},  /* -7 */
{0, 0, 6},  /* -6 */
{0, 0, 5},  /* -5 */
{0, 0, 4},  /* -4 */
{0, 0, 3},  /* -3 */
{0, 0, 2},  /* -2 */
{0, 0, 1},  /* -1 */
{0, 0, 0},  /* 0 */
{0, 0, -1}, /* 1 */
{0, 0, -2}, /* 2 */
{0, 0, -3}, /* 3 */
{0, 0, -4}, /* 4 */
{0, 0, -5}, /* 5 */
{0, 0, -6}, /* 6 */
```

```
{0, 0, -7}, /* 7 */
{0, 0, -8}, /* 8 */
{0, 0, -9}, /* 9 */
{0, 0, -10},       /* 10 */
{0, 0, -11},       /* 11 */
{0, 0, -12},       /* 12 */
{0, 0, -13},       /* 13 */
{0, 0, -14},       /* 14 */
{0, 0, -15},       /* 15 */
{0, 0, -16},       /* 16 */
{0, 0, -17},       /* 17 */
{0, 0, -18},       /* 18 */
{0, 0, -19},       /* 19 */
{0, 0, -20},       /* 20 */
{0, 0, -21},       /* 21 */
{2, 0, 20}, /* 22 */
{2, 0, 19}, /* 23 */
{2, 0, 18}, /* 24 */
{2, 0, 17}, /* 25 */
{2, 0, 16}, /* 26 */
{2, 0, 15}, /* 27 */
{2, 0, 14}, /* 28 */
{2, 0, 13}, /* 29 */
{2, 0, 12}, /* 30 */
{2, 0, 11}, /* 31 */
{2, 0, 10}, /* 32 */
{2, 0, 9},  /* 33 */
{2, 0, 8},  /* 34 */
{2, 0, 7},  /* 35 */
{2, 0, 6},  /* 36 */
{2, 0, 5},  /* 37 */
{2, 0, 4},  /* 38 */
{2, 0, 3},  /* 39 */
{2, 0, 2},  /* 40 */
{2, 0, 1},  /* 41 */
{2, 0, 0},  /* 42 */
{2, 0, -1}, /* 43 */
{2, 0, -2}, /* 44 */
{2, 0, -3}, /* 45 */
{2, 0, -4}, /* 46 */
{2, 0, -5}, /* 47 */
{2, 0, -6}, /* 48 */
{2, 0, -7}, /* 49 */
{2, 0, -8}, /* 50 */
{2, 0, -9}, /* 51 */
{2, 0, -10},       /* 52 */
{3, 0, 10}, /* 53 */
{3, 0, 9},  /* 54 */
{3, 0, 8},  /* 55 */
{3, 0, 7},  /* 56 */
{3, 0, 6},  /* 57 */
{3, 0, 5},  /* 58 */
{3, 0, 4},  /* 59 */
{3, 0, 3},  /* 60 */
{3, 0, 2},  /* 61 */
{3, 0, 1},  /* 62 */
{3, 0, 0},  /* 63 */
```

```
{3, 0, -1}, /* 64 */
{3, 0, -2}, /* 65 */
{3, 0, -3}, /* 66 */
{3, 0, -4}, /* 67 */
{3, 0, -5}, /* 68 */
{3, 0, -6}, /* 69 */
{3, 0, -7}, /* 70 */
{3, 0, -8}, /* 71 */
{3, 0, -9}, /* 72 */
{3, 0, -10},       /* 73 */
{3, 1, 10}, /* 74 */
{3, 1, 9},  /* 75 */
{3, 1, 8},  /* 76 */
{3, 1, 7},  /* 77 */
{3, 1, 6},  /* 78 */
{3, 1, 5},  /* 79 */
{3, 1, 4},  /* 80 */
{3, 1, 3},  /* 81 */
{3, 1, 2},  /* 82 */
{3, 1, 1},  /* 83 */
{3, 1, 0},  /* 84 */
{3, 1, -1}, /* 85 */
{3, 1, -2}, /* 86 */
{3, 1, -3}, /* 87 */
{3, 1, -4}, /* 88 */
{3, 1, -5}, /* 89 */
{3, 1, -6}, /* 90 */
{3, 1, -7}, /* 91 */
{3, 1, -8}, /* 92 */
{3, 1, -9}, /* 93 */
{3, 1, -10},       /* 94 */
{3, 2, 10}, /* 95 */
{3, 2, 9},  /* 96 */
{3, 2, 8},  /* 97 */
{3, 2, 7},  /* 98 */
{3, 2, 6},  /* 99 */
{3, 2, 5},  /* 100 */
{3, 2, 4},  /* 101 */
{3, 2, 3},  /* 102 */
{3, 2, 2},  /* 103 */
{3, 2, 1},  /* 104 */
{3, 2, 0},  /* 105 */
{3, 2, -1}, /* 106 */
{3, 2, -2}, /* 107 */
{3, 2, -3}, /* 108 */
{3, 2, -4}, /* 109 */
{3, 2, -5}, /* 110 */
{3, 2, -6}, /* 111 */
{3, 2, -7}, /* 112 */
{3, 2, -8}, /* 113 */
{3, 2, -9}, /* 114 */
{3, 2, -10},       /* 115 */
{3, 3, 10}, /* 116 */
{3, 3, 9},  /* 117 */
{3, 3, 8},  /* 118 */
{3, 3, 7},  /* 119 */
{3, 3, 6},  /* 120 */
```

```
{3, 3, 5},  /* 121 */
{3, 3, 4},  /* 122 */
{3, 3, 3},  /* 123 */
{3, 3, 2},  /* 124 */
{3, 3, 1},  /* 125 */
{3, 3, 0},  /* 126 */
{3, 3, -1}, /* 127 */
{3, 3, -2}, /* 128 */
{3, 3, -3}, /* 129 */
{3, 3, -4}, /* 130 */
{3, 3, -5}, /* 131 */
{3, 3, -6}, /* 132 */
{3, 3, -7}, /* 133 */
{3, 3, -8}, /* 134 */
{3, 3, -9}, /* 135 */
{3, 3, -10},       /* 136 */
{3, 3, -11},       /* 137 */
{3, 3, -12},       /* 138 */
{3, 3, -13},       /* 139 */
{3, 3, -14},       /* 140 */
{3, 3, -15},       /* 141 */
{3, 3, -16},       /* 142 */
{3, 3, -17},       /* 143 */
{3, 3, -18},       /* 144 */
{3, 3, -19},       /* 145 */
{3, 3, -20},       /* 146 */
{3, 3, -21},       /* 147 */
{3, 3, -22},       /* 148 */
{3, 3, -23},       /* 149 */
};
```